FACULTÉ
DES SCIENCES

UNIVERSITÉ LIBRE DE BRUXELLES

Some Practical Aspects of Lattice-based Cryptography

Thesis submitted by François GÉRARD
in fulfilment of the requirements of the PhD Degree in Sciences
Academic year 2019-2020

Supervisor: Olivier MARKOWITCH
QUALSEC (Quality and Security of Information Systems)

Thesis jury :

Joppe BOS (NXP Semiconductors)
Jean-Michel DRICOT (Université libre de Bruxelles, Secrétaire)
Olivier MARKOWITCH (Université libre de Bruxelles)
Christophe PETIT (University of Birmingham)
Yves ROGGEMAN (Université libre de Bruxelles, Président)
Gilles VAN ASSCHE (STMicroelectronics)

Acknowledgements

The journey towards the writing of this PhD thesis has been filled with opportunities to meet new people. Obviously, trying to exhaustively refer to each of them would unfortunately imply some unintentional omissions. However, there are people without whom it would not even have started. Hence I would like start by showing my gratitude to Olivier and Yves who gave me the opportunity to start the journey. Outside of science, and even though Olivier participated sporadically, Yves deserves a special mention for being present to most of the lunch events organized in the past years. There are also people without whom it could not be completed: my thesis committee. Thanks to Jean-Michel for accepting to hear me once again presenting my research work. Thanks to Joppe who I rarely met but spent countless hours studying papers he authored in the world of lattice-based cryptography. Thanks to Gilles who not only accepted to be part of my committee but also did a great job renewing the cryptography course of ULB last year. And finally, thanks to Christophe who I did not know until very recently and kindly accepted my invitation even though the topics covered in this document are different from what he usually works on. But of course, between the start and the end, there are all the people that I worked with on academic papers that are the backbone of this document. Thanks to all my coauthors. Since I was not only a PhD student but also a teaching assistant at ULB, I would also like to thank all my colleagues. First and foremost Keno, Stéphane, Liran and Nikita which are the people I spent most of my time and, more importantly, all of my breaks with, but also all the others teaching assistants, especially the ones present at lunchtime every Thursday. I also thank all the researchers, professors and post-docs from the department, as well as the secretaries who worked to organize the teaching activities. Naturally, I am also grateful to my family and friends because there also a life outside of the university. Finally, the last thanks go to Lila who joined me halfway through the PhD but has been steadily by my side and supported me ever since.

Contents

Chapter 1

Abstract and Organization

The topic of this thesis is post-quantum cryptography, that is to say, cryptography resisting classical and quantum cryptanalysis. This topic gained a lot of traction in the past few years, helped by the establishment of a standardization process by the American National Institute of Standards and Technology (NIST). While the design and security properties of post-quantum cryptosystems have been studied for a longer time, the call for proposals from NIST at the end of 2016 resulted in a portfolio of concrete instanciations for encryption and signature schemes. There exists several families of assumptions enabling post-quantumness. In this thesis, we focus exclusively on schemes basing their security on the hardness of computational problems over mathematical objects called lattices. More specifically, our main concern will be the practical aspects of lattice-based cryptography, that is to say we care about efficiency and security once the cryptosystem is used in the real world. Let us be more specific on the different topics that will be covered.

Efficient implementations

Using a cryptographic algorithm in practice naturally requires to implement it on a specific platform. Since cryptography comes, as a necessary overhead, on top of the functionalities of a user service it should not slow it down significantly and should also be deployable at a reasonable cost. Therefore, finding fast algorithms computing the mathematical transformations specified by a cryptographic scheme and implementing them efficiently is the first milestone to practicality. This is particularly true with public-key primitives requiering to perform computations on algebraic objects. Chapter 3 will discuss this topic by presenting a research work about efficient implementations of algorithms manipulating polynomials in the ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ on a popular microcontroller. This Chapter will be supported by our paper "Cortex-M4 Optimizations for {R,M}LWE Schemes" [6].

Side-Channel countermeasures

Side-Channel Attacks exploit weaknesses in implementations and/or devices performing computations. In Chapter 4, we will discuss power analysis, a type of attack measuring the power consumption of a device while it runs a cryptographic algorithm and trying to extract some secret data from those measurements afterward. It is clear that this type of attack is only relevant once the cryptosystem is used in practice in specific environments. However, general countermeasures can be designed independently of the real world usage. Among them, we will present a popular technique called masking that will be applied to a signature scheme candidate for standardization in the NIST project. Power analysis will

be introduced via our paper 'Breaking Kalyna 128/128 with Power Attacks [53]' and the post-quantum relevant part is the masking scheme of our paper "An Efficient and Provable Masked Implementation of qTESLA" [61].

Efficient design

One looking for efficiency will of course try to choose the more efficient cryptosystem available to oneself. However, when several security goals have to be met, stacking multiple primitives might incur an unforeseen overhead. Therefore, designing primitives streamlined to cover different security requirements at the same time can result in an efficiency boost. In the lighter Chapter 5, we will present a primitive called signcryption. The goal of a signcryption scheme is to offer the functionalities (and of course security) of both a signature and an encryption scheme at the same time. The Chapter presents a post-quantum lattice-based signcryption scheme which is the result of our paper "SETLA: Signature and Encryption from Lattices" [59].

Organization of the Main Chapters

The three main Chapters (3, 4 and 5) will start with a section called "Preamble" in which I will talk in first person to explain the events or context that led me to work on the covered topic. The text will then switch to scientific writing using the pronoun "we" to give specific introduction and preliminaries, followed by a section presenting contributions. Since most contributions made their way to the literature, the title of the corresponding sections indicate in which conference/journal they were published. Finally, Chapters are closed by a "Thoughts and future works" section that switch back to first person. The goal of this last section is to look back at the accomplished work with a critical eye and discuss possible flaws or extensions.

List of Publications

This document will cover the material of the four following publications:

Breaking Kalyna 128/128 with Power Attacks (SPACE 2016)
S.Fernandes Medeiros, F.Gérard, N.Veshchikov, L.Lerman and O. Markowitch [53]

SETLA: Signature and Encryption from Lattices (CANS 2018)
F.Gérard and K. Merckx [59]

An Efficient and Provable Masked Implementation of qTESLA (CARDIS 2019)
F.Gérard and M. Rossi [61]

Cortex-M4 Optimizations for {R,M}LWE Schemes (CHES 2020)
E. Alkim, Y. A. Bilgin, M. Cenk and F.Gérard [6]

During my work as a PhD candidate, I cosigned one more publication

Fully Homomorphic Distributed Identity-based Encryption Resilient to Continual Auxiliary Input Leakage (SECRYPT 2018)
F.Gérard, V. Kuchta, R. Anand Sahu,G. Sharma and O. Markowitch [58]

that will not be discussed in this manuscript.

Mathematical Notations

The mathematical notations used in the text are fairly standard and should be easily understandable without help. Non-standard notations will explained at the time they are needed.

Sets of numbers. We use $\mathbb{Z}$ and $\mathbb{R}$ to denote the integers and the real numbers. Sets are written with the usual bracket notation $\{a, b, c, \dots\}$. The integers between $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$ will be expressed using the interval notation $[a, b]$ or the set notation $\{a, a+1, \dots, b\}$.

Modular arithmetic. The short notation $\mathbb{Z}_q$ is used for $\mathbb{Z}/q\mathbb{Z}$. It will be mostly represented with the classical representatives set $\{0, 1, \dots, q-1\}$ or the centered representative set $\{-\lfloor q/2 \rfloor, \dots, \lfloor (q-1)/2 \rfloor\}$. We write mod q to denote either an equivalence modulo q or, during computations, an explicit reduction modulo q in whatever set of representatives is used. The context should clearly imply which one it is. An explicit reduction to the set of centered representative modulo q is written $\bmod^{\pm} q$.

Rings and fields. The main rings used are $\mathbb{Z}_q$ and the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle f \rangle$ for a given irreducible polynomial f which will almost always be $X^n + 1$, with n a power of two, making it a cyclotomic polynomial. Since q will often be a prime, $\mathbb{Z}_q$ is often a field but this property will not be used. The rare times an explicit k elements field is needed are noted as $GF(k)$. In two Chapters, the important notation (that will be recalled in the text) $\mathcal{R}_{q,[B]}$ appears. It means the set of polynomials in $\mathcal{R}_q$ with coefficients in $[-B, B]$ for $B < q/2$.

Elements, vectors and matrices. Elements of groups, rings or fields are written using lowercase letters, e.g. $a \in \mathbb{Z}$, vectors are written using bold lowercase letters , e.g. $\mathbf{v} \in \mathbb{Z}^n$, and matrices are written using bold uppercase letters, e.g. $\mathbf{M} \in \mathbb{Z}^{n \times m}$. The notation $\|\mathbf{v}\|_p$ denotes the ℓ_p-norm. When the subscript is omitted, we mean the Euclidean norm $\|\cdot\|_2$. Be careful that we will work most of the time with polynomials that are elements in a polynomial ring. They are therefore written with lowercase letters but are still multidimensional objects and are sometimes interpreted as a vector of their coefficients. The interpretation should be clear from context.

Sampling and distributions. When a random value v is drawn from a distribution D, we write $v \xleftarrow{r} D$. The notation is extended to randomized algorithms, e.g. for an algorithm A, $x \xleftarrow{r} \mathsf{A}()$ means that A is run on some implicit randomness and returns x. When the notation is used with a set (this can be the set of elements of an algebraic structure) instead of a distribution or an algorithm, we mean the uniform distribution over this set. When we want to refer explicitly to the uniform distribution over the set S, we sometimes write $\mathcal{U}(S)$.

Chapter 2

Introduction and Preliminaries

2.1 Introduction

If I had to discuss the most fundamental lifestyle change between my youth in the 90s and the writing of this thesis in 2020, I would, without hesitation, point out the usage of digital technologies. This feeling is confirmed by a quick glance at the market capitalization of large public companies: Apple, Microsoft, Google, Amazon, Facebook and Alibaba are currently shining somewhere at the top. The rise of those companies is correlated with the soaring success of the Internet. Barely existent at the time of my birth, it is now present in almost all households of developed country and an everyday tool for a large part of their population, used for entertainment, services or communication.

At the core of those changes lie the concept of information exchange and processing. While it is not new that people are communicating, storing and interpreting data, the means to do it drastically changed through history. From an ancient library filled with books to a modern data center, from Philippides running between Marathon and Athens to high speed data flow through a network, from Byzantine generals reading attack orders to a machine learning algorithm trying to beat the best Starcraft players. Those modifications in speed and magnitude enabled the creation of a real business around information where data are as valuable as tangible goods.

Naturally, higher demand for data processing and collection leads to higher demand for data security. Unfortunately, the term *data security* is quite vague and encompass a lot of informal notions. Is a message written on a paper note in my pocket while I am driving my car securely transmitted? Is an IP packet traveling in an unattainable underground wire securely transmitted? Those questions can solely be answered if one provides a clear and unambiguous definition of security.

The scientific field studying the core concepts of secure data transmission is called *cryptography*. While it has a long history as an "art", the formal, rigorous, scientific version of cryptography really developed itself during and after the second world war and grew significantly in parallel with the expansion of information technologies such as phones, computers, and their global evolution, internet connected devices.

Giving a rigorous framework to security requires to formally describe what are the possible threats to the considered system. In cryptography, those threats are often called the adversary. Thus, one way to give a formal notion of security is to define an adversary by putting hypothesis on its capabilities along with tasks that should be unachievable for him, used to model desirable security properties. Security is then argued by formally showing that solving those tasks would contradict the hypothesis. For instance, the adver-

sary, modelled as an algorithm, is (almost) always assumed to run in polynomial time (in some security parameter). If we can then show that breaking a desired security property implies solving a hard [1] problem, we know that the property will hold under the assumption that the problem is indeed hard. This is actually the well known concept of reduction from computability theory. One of the major challenges to migrate cryptography from scientific papers to concrete information systems is to actually make hypothesis on the adversary that translate well into the real world.

The work of this thesis lies in the field of *post-quantum cryptography*, which is a subset of cryptography in which the adversary is assumed to be also equipped with a quantum computer. This assumption changes the landscape of the public-key side of cryptography. Indeed, some computational problems widely used in the asymmetric setting (factoring and the discrete logarithm) that are believed to be outside of P for a classical adversary become tractable for a quantum adversary. Note that the assumption that the (set of algorithms used by the) adversary runs in polynomial time still holds, the gap between classical and post-quantum cryptography lies in the fact that the set of problems solvable in polynomial time by a quantum computer is currently believed to be a proper superset of P.

The goal of this thesis is to discuss the *practicals aspects* of post-quantum cryptography and, more concretely, schemes built using hard problems on lattices. The term "practical" is very broad and has different meaning in different situations but is used here has a mean to say "ways to bring lattice-based cryptography from theory to the real world". More specifically, the core of the document will be split in three main parts: high-speed implementations, counter-measures against physical attacks, and application of existing efficient designs. Those topics illustrate the various issues underlying the quest to make post-quantum cryptography usable in practice.

2.2 Cryptography

Our starting point for this brief introduction to cryptography will be the definition of the popular "old but good" *Handbook of Applied Cryptography* [88]:

> *Cryptography* is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.

For the purpose of this thesis, we would like to update this quote in the following way:

> *Cryptography* is the study of mathematical techniques**, and the issues arising from their instanciation in real-life applications,** related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.

The reason for this modification is that, over the years, studying practicals issues like physical attacks or efficient implementations has been a more and more common activity in the academic literature. This phenomenon is due to the tremendous growth and diversification of electronic devices between the birth of computer science as a branch of mathematics and today. Having researchers, as neutral observers, comparing, in a

[1] By hard, we mean computationally hard, that is to say out of reach in practice!

scientific setting, performances of various architectures or products from different manufacturers is now a common practice. Furthermore, cryptographic implementations are often (especially on the asymmetric side) computations involving mathematical objects from number theory and algebra, which are topics fitting well the type of people who live in the academic world.

Returning to our definition, we see the word "mathematical" appearing quickly. The reason behind it is that cryptography primarily deals with information and computation from an abstract point of view. Quite often, cryptographic schemes are designed solely from a mathematical point of view together and complexity theory notions are used to argue hardness. The issues that may arise when implementing it in a real-life device are often studied afterward. We will see in Chapter 4 a concrete example in which this approach may be noxious when considering physical attacks. Nevertheless, it has a crucial advantage, which is that, outside of basic assumptions on computational capabilities, cryptographic algorithms do not depend on the machine used to run them. This means a paradigm shift in the way we build computers would not invalid the design of the schemes. Naturally, there will always be design choices taking into consideration real-life variables. For example, the famous Advanced Encryption Standard comes with three possible key lengths: 128, 192, and 256, which are values chosen after estimated upper bounds on the brute-forcing power of realistic future adversaries. Further come the four following notions:

- Confidentiality: The data transferred during communication should remain secret to an unauthorized party (the adversary). The adversary can see that some data is communicated but the content should be kept unknown to him. The basic cryptographic primitive used to ensure confidentiality is called *encryption*.

- Data integrity: The data received at the endpoint should be the same as the data sent at the starting point. Here, we usually do not segregate malicious and unintentional data modification. An unreliable channel might be seen as an adversary performing random modifications on the data. The basic cryptographic building block used for data integrity is called *hash function* but more advanced constructions like *message authentication codes (MAC)* and *signatures* also ensure integrity.

- Entity authentication: Entities involved in a communication should be able to prove their identity to each other. This can be achieved using *authentication protocols*, *MACs* or *signatures*.

- Data origin authentication: The receiver of some data should be able to identify the source. More precisely, it should be possible to verify that data claimed to be sent by an entity E is indeed coming from E and not from an impersonator. Again, *MACs* and *signatures* are used in this case.

The reason why the three last properties are ensured with common cryptographic algorithms is that they are mostly needed together in practice. Indeed, if undetected modifications to the message can be performed, efforts to validate sender's identity are somewhat worthless. Similarly, if genuine senders cannot be distinguished from impersonators, the adversary does not need to modify existing messages. Conversely, confidentiality can be enabled or disabled without affecting other properties.

The four security notions discussed above form the basis of cryptography. It is hard to imagine a real-life context in which none of them is needed in a sensible (i.e. needing

security) environment. Actually, these are needed so regularly that specialized cryptographic primitives encompassing all of them were designed: *authenticated encryption* and *signcryption*, which is the topic of Chapter 5. Nevertheless, cryptography went beyond these basic notions with primitives offering richer properties. We can, for example, cite *homomorphic encryption*, with which it is possible to compute on encrypted data without revealing them or *ring signatures*, offering the possibility to sign a message under an aggregate of identities. These advanced constructions are also studied in a post-quantum setting and homomorphic encryption is especially strongly tied to lattice-based cryptography. However, the main topics of this thesis are the simpler constructions of encryption and signature as they are the core of all cryptography and, hence, naturally studied and put in place first.

2.2.1 Encryption

Probably the most well-known application, encryption aims to conceal the content of a message to an adversary eavesdropping the communication channel. The situation is the following: a sender (a.k.a. Alice) wants to send a message to a receiver (a.k.a Bob) but every data transiting between Alice and Bob can be read by an eavesdropper (a.k.a Eve). Alice will then encrypt the message into a *random looking* piece of data called the *ciphertext* and send it on the channel. At the other endpoint, Bob will decrypt the ciphertext to retrieve the original message. Eve will only see the random looking ciphertext on the channel and will not learn anything about the message. Of course, it should be impossible for Eve to decrypt, which means that Bob has some capabilities that Eve does not have.

No introduction to encryption would be adequate without the infamous and over used toy example of Caesar's cipher. In this cipher, Alice and Bob communicate with messages that are words over the 26 letters of the Latin alphabet. Before communicating, they will first agree on a *secret value* k called the key. Then, when Alice wants to send a message to Bob, she shifts every letter of k positions in the alphabet (if some letter goes past Z, it loops back to A) and send the result on the channel. When Bob receives the ciphertext, he applies the shift in reverse to retrieve the original message. The assumption is that Eve cannot read the message because she does not know the secret shifting value k.

While this cipher provides no security for a myriad of reasons (the first one being that there is only 26 possible shifts, which is a piece of cake to brute-force for a modern computer), it is of great helpfulness to illustrate a formal abstraction of an encryption scheme.

Definition 1. *(Encryption) An encryption scheme is a tuple* $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $\mathcal{M}, \mathcal{C}, \mathcal{K}$ *are sets called the messages, ciphertexts and keys spaces.*
- $\mathsf{Gen} : \lambda \to \mathcal{K}$ *is a key generation function taking as input a security parameter and outputting a key.*
- $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ *is a function taking as inputs a key and a message and outputting a ciphertext called the encryption procedure.*
- $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ *is a function taking as inputs a key and a ciphertext and outputting a message called the decryption procedure.*

An alternative and elegant definition is to see Enc and Dec as *families* of functions indexed by elements of $\mathcal{K}$.

It is generally assumed that mapping the real-life message to the message space is a trivial operation. For example, in the case of Caesar's cipher, the encryption and decryption procedures are often described as $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$, $\mathsf{Enc} : k, m[i] \mapsto c[i] = m[i] + k \mod 26$ and $\mathsf{Dec} : k, c[i] \mapsto m[i] = c[i] - k \mod 26$ which assumes that the message is a sequence of numbers in $\mathbb{Z}_{26}$ and the trivial map sending each letter to its index in the alphabet is used to encode the real-life message. As encryption schemes often have message spaces that are $\{0,1\}^*$ or sets with elements easy to encode with integers, the usual binary representation used by computers fits well for digital applications.

Asymmetric cryptography

Assuming our example above replaces Caesar's cipher by a modern and secure encryption method, it illustrates the importance of the knowledge of the key as it is the only difference between Bob and Eve[2]. This raises an important question: how can Alice and Bob agree on a shared key if they do not have a secure communication channel yet? The answer could simply be that having a shared secret is an assumption of the system and this should be done by another mean, e.g. meeting in person, but this is not satisfactory on a large scale, especially in a world where data is traveling lightning fast across the globe. The solution comes from what is called *asymmetric (mainly called public-key)* cryptography. Whereas *symmetric* cryptography has algorithms using identical secret keys shared among participants, asymmetric cryptography associates two different keys to each participant, a private one that is kept secret and a public one that is known to the adversary. A public-key encryption scheme would be formalized as follows:

Definition 2. *(Public-key (asymmetric) Encryption) A public-key encryption scheme is a tuple* $(\mathcal{M}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *in which:*

- $\mathcal{M}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}$ *are sets called the messages, ciphertexts, public keys and secret keys*[3] *spaces.*
- $\mathsf{Gen} : \lambda \to \mathcal{SK} \times \mathcal{PK}$ *is key generation function taking as input a security parameter and outputting a key pair.*
- $\mathsf{Enc} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C}$ *is a function taking as inputs a public key and a message and outputting a ciphertext.*
- $\mathsf{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M}$ *is a function taking as inputs a secret key and a ciphertext and outputting a message.*

In this setting, when Alice wants to secretly send a message m to Bob, they undergo the following steps: Bob first run the key generation algorithm Gen to get a key pair (sk, pk) and announces publicly pk. Alice then computes the ciphertext $c = \mathsf{Enc}(pk, m)$ and sends it on the channel. Finally Bob computes $m = \mathsf{Dec}(sk, c)$ and retrieve the message. The main assumption is that the message cannot be computed from the ciphertext without knowing the secret key sk. However, this naive property is not sufficient and

[2] Actually, cryptographic schemes ensure their security *solely* by the secrecy of the key(s), all the algorithms and parameters used are public.

[3] While the adjective private fits better the setting of asymmetric cryptography, private keys are often called secret keys in order to clear confusion concerning the shortened form "pk".

more advanced required security notions will be presented in Section 2.2.4. While the general idea is the same as in the previous section, the notable difference here is that no prior secret information was shared between Alice and Bob, she only needs to know his public key. An issue that will not be discussed here but which has a *tremendous* importance in practice is that it might be hard to associate public keys to real-life identities. In the scenario we just described, when Alice sees the announce of the public key from Bob, she needs to ensure that it is not an impersonator claiming to be Bob.

At first sight, it seems that symmetric encryption should be superseded by public-key encryption as it offers the same service without a need for key distribution. While it is a theoretically acceptable argument, in practice, public-key algorithms are using substantially more resources than their symmetric counterpart. Therefore, it is common to operate a hybrid mode in which asymmetric cryptography is firstly used to share a secret symmetric key and all subsequent communications are secured with symmetric encryption.

2.2.2 Diffie-Hellman Key Exchange

Since the public-key part of a hybrid mode is solely used to establish a shared key between participants, specialized cryptographic algorithms having this only purpose were proposed. The most popular of them, due to Diffie and Hellman [46], is actually seen as the first example of concrete public-key cryptography. Its security is related to the discrete logarithm problem:

Definition 3. *(discrete logarithm problem) Let G be a finite cyclic group equipped with a multiplicative law $\circ$ and g one of its generator. Given an element $y \in G$, find the integer x such that $g^x = \underbrace{g \circ g \ldots g}_{x \text{ times}} = y$.*

The Diffie-Hellman key exchange works in any group but would not be secure in groups in which the discrete logarithm problem is tractable. Its first instanciation was using the multiplicative subgroup of a finite prime field of p elements, which is usually denoted $\mathbb{Z}_p^*$. More recent constructions use groups of points on elliptic curves over finite fields but are out of scope of this document as the conceptual view of the key exchange is kept the same and they do not offer post-quantum security either[4].

The key exchange is depicted as a protocol in Figure 2.1. Working in a cyclic group of order q with generator g, the concept is quite simple: first, Alice and Bob respectively pick random numbers a and b in (canonical representatives of elements of) $\mathbb{Z}_q$ which can be seen as private keys in the asymmetric framework. Then, Alice publishes $pk_a = g^a$ and Bob publishes $pk_b = g^b$ which can be seen as public keys. To establish the shared secret, Alice will rise the public key of Bob to its private key and vice versa for Bob. They both end up with the shared value $k = g^{ab}$.

The adversary eavesdropping on the channel only sees the values g^a and g^b. If the discrete logarithm problem is hard in the group used to perform the protocol, the adversary cannot retrieve neither a nor b. Nonetheless, it should be carefully noted that their goal is not to retrieve a or b but to find the shared key g^{ab}. This is why claiming that the security is based on the discrete logarithm is inaccurate, the actual underlying problem is simply called the (computational) Diffie-Hellman problem (CDHP).

[4]Some elliptic curves are used in post-quantum cryptography but not with the discrete logarithm assumption.

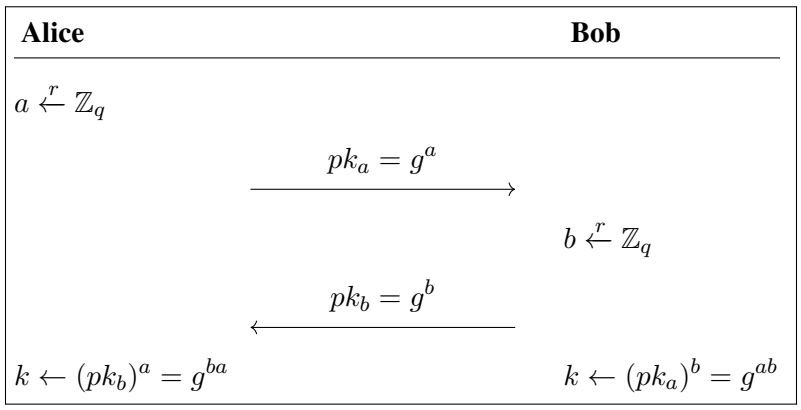

Figure 2.1: Diffie-Hellman key exchange in a group G of order q with generator g. We note that the order of the messages is not important here, Bob can send pk_b before receiving pk_a.

Definition 4. *(Computational Diffie-Hellman problem) Let G be a finite group equipped with a multiplicative law and g one of its generator. Given two elements g^x and g^y, find g^{xy}.*

Nevertheless, the most straightforward way to solve the CDHP is to actually compute the discrete logarithm on one of the two inputs to get x or y and use it together with the other input to compute g^{xy}. This is why Diffie-Hellman is often mistakenly described as a protocol basing its security on the discrete logarithm.

2.2.3 From Key Exchange to Encryption: the ElGamal Cryptosystem

Assuming we have access to a symmetric encryption scheme, the Diffie-Hellman key exchange is sufficient to create a public-key encryption scheme. Indeed, Alice can pick a, compute the shared key $k = pk_b^a$, encrypt her message with the symmetric scheme under the key k and send to Bob the symmetric encryption together with g^a. We will soon describe such a cipher due to ElGamal [55].

But first, we need to introduce the concept of perfect secrecy and give a cipher that offers this property called the (generalized) one-time pad. This cipher will be implicitly used in the construction of ElGamal.

Definition 5. *(perfect secrecy) A cipher achieves the perfect secrecy property if, for $c = \mathsf{Enc}(m)$, the probability of the message being m is unchanged by learning the ciphertext c. That is to say $\forall m, P[M = m] = P[M = m|C = c]$.*

Intuitively, this captures the fact that an adversary seeing the ciphertext will not gain any information on the plaintext. The strength of this approach is that the security is not based on a computational assumption but rather on an information-theoretic argument, which means that the adversary can be assumed to have access to an unlimited amount of computational resources.

Generalized one-time pad

Now, let us discuss a really simple symmetric encryption scheme that we will call the Generalized one-time pad (GOTP).

Definition 6. *(Generalized one-time pad)*
Let $(G, \circ)$ be a finite group.

- $\mathcal{M} = \mathcal{K} = \mathcal{C} = G$.
- $\mathsf{Enc} : (k, m) \mapsto c = k \circ m$
- $\mathsf{Dec} : (k, c) \mapsto m = k^{-1} \circ c$

Encryption is simply given by applying the group law operator on the message and the key and decryption reverse this operation using the existence of an inverse for every k.

Proposition 1. *The cipher in Definition 6 offers perfect secrecy under the assumption that the key is drawn from a perfectly uniform distribution.*

Proof. Let I be an arbitrary set indexing the elements of G. The probability of having a ciphertext c is given by

$$P[C = c] = \sum_{i \in I} P[M = m_i] \cdot P[K = c \circ m_i^{-1}].$$

Since the key is drawn from a uniform distribution, we have

$$P[C = c] = \sum_{i \in I} P[M = m_i] \cdot \frac{1}{|G|} = \frac{1}{|G|}.$$

Using the fact that

$$P[C = c | M = m] = P[K = c \circ m^{-1}],$$

we have by the same argument that

$$P[C = c | M = m] = \frac{1}{|G|} = P[C = c].$$

By Bayes,

$$P[M = m | C = c] = \frac{P[C = c | M = m] \cdot P[M = m]}{P[C = c]} = 1 \cdot P[M = m] = P[M = m]$$

.

□

We note that Ceasar's cipher is similar to the GOTP using $(G, \circ) = (\mathbb{Z}_{26}, +)$. Its lack of security actually comes from the fact that the same group element k is used to encrypt multiple elements (letters) in $\mathbb{Z}_{26}$, this is why this cipher is called "one-time" pad, the key must be used only once per plaintext (group element).

Public parameters:

- cyclic group G of order q generated by g
- encoding function H mapping messages to G

Decryption key: $sk \xleftarrow{r} \mathbb{Z}_q$
Encryption key: $pk \leftarrow g^{sk}$

ElGamal Encrypt(pk, m):

1: $r \xleftarrow{r} \mathbb{Z}_q$
2: $c_1 \leftarrow g^r$
3: $c_2 \leftarrow pk^r \circ \mathsf{H}(m)$
4: **return** c_1, c_2

ElGamal Decrypt(c_1, c_2, sk):

1: $m \leftarrow c_2 \circ c_1^{-sk}$
2: $m \leftarrow \mathsf{H}(m)$
3: **return** m

Figure 2.2: ElGamal Encryption

The usual one-time pad

The usual one-time pad (OTP) denotes the cipher $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^*$, $\mathsf{Enc}(k, m) = k \oplus m$ and $\mathsf{Dec}(k, c) = k \oplus c$ in which encryption and decryption can be computed using the same circuit with a simple XOR operation between inputs. This is a variant of the cipher in Definition 6 with $(G, \circ) = (\mathbb{Z}_2, +)$ which runs multiple instances in parallel. The reason for its attractiveness is twofold: representation of messages as bitstrings comes for free in digital communications and the $+$ operation in $\mathbb{Z}_2$ is self-inverse and trivial to compute. While this encryption scheme is very simple and very fast, its obvious drawback is that the key and the message should have the same size, which is unpractical in many scenarios, e.g. encrypting a whole hard drive. Actually, the one-time pad is more realistically seen as a tool used to construct other cryptosystems instead of a whole cipher. This tool could be lousy stated as "a random bitstring can be used to perfectly hide another bitstring of the same size". The most straightforward construction based on the OTP is called a stream cipher. The idea is to create a key expansion algorithm taking as input a small key and outputting an arbitrary large keystream. To encrypt, use the key expansion algorithm to get a keystream as long as the message and then simply XOR both of them. The security of such a construction depends on the pseudo-randomness of the keystream, that is to say, on the quality of the key expansion algorithm.

The ElGamal cryptosystem

Let us now describe the ElGamal encryption scheme. It was proposed in 1985 by Taher Elgamal [55] in a paper discussing public-key cryptosystems. It results from the observation that the Diffie-Hellman protocol can be performed in a non-interactive way. Let us say Bob as computed a keypair $(sk, pk = g^{sk})$ and publicly announced pk. When Alice wants to encrypt a message for him, she picks a secret value r associated with a public value[5] $c_1 = g^r$ and computes her side of the Diffie-Hellman protocol to get an ephemeral key pk^r. This key is then used to encrypt the message encoded as a ring element using the Generalized one-time pad. Decryption is straightforward, Bob computes

[5] which correspond to the pair (a, pk_a) in Figure 2.1.

his side of the DH key exchange and trivially decrypts. Figure 2.2 formally describes the cryptosystem. The security of this encryption scheme is naturally closely tied to the one of the Diffie-Hellman key exchange but we shall be more precise with security notions when discussing encryption schemes. We will now describe a popular way to formalize security using a game between an adversary and a challenger.

2.2.4 Semantic Security

Since the word "security" has different meaning depending on the context, it is widespread to express it in terms of a game between an adversary and a challenger in which the capability of each entity is clearly defined. Answering the question of whether the game correspond to the real-life situation in which the cryptosystem is used is a responsibility of the user.

The most popular and essential security notion is semantic security, which basically states that amount of information on the plaintext the adversary can learn from the ciphertext should be negligible. It is formalized by the property of indistinguishability under chosen plaintext attack (IND-CPA) described by the following game:

Definition 7. *(IND-CPA game) Let* $\mathsf{PKE} = (\mathcal{M}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme* [6] *and* λ *a security parameter. The challenger and the adversary, both modeled as algorithms running in polynomial time in* λ *play the following game:*

1. *The challenger runs the key generation algorithm to get a key pair* $(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$ *and publicly announces* pk*.*
2. *The adversary arbitrarily chooses two messages of the same size* m_0, m_1 *and transmits them to the challenger.*
3. *The challenger picks a random bit* $b \xleftarrow{r} \{0, 1\}$*, sets* $c \leftarrow \mathsf{Enc}(pk, m_b)$ *and returns* c *to the adversary.*
4. *The adversary outputs a bit* b' *and wins the game if* $b' = b$*.*

The encryption scheme PKE *is IND-CPA iff the advantage of the adversary* $|Pr[b = b'] - \frac{1}{2}|$ *is a negligible function in* λ*.*

The above security notion is aimed toward *passive adversaries*, that is to say an adversary simply observing the channel. We now introduce the stronger notion of indistinguishability under chosen ciphertext attack (IND-CCA) in which the adversary also has access to a decryption oracle helping him to distinguish encrypted messages. The decryption oracle is a black box function capable of decrypting any message, it is used to model the fact that in a real-life scenario, the adversary might have access to the decryption of ciphertexts different from the one they are attacking.

Definition 8. *(IND-CCA game) Let* $\mathsf{PKE} = (\mathcal{M}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme and* λ *a security parameter. The challenger and the adversary, both modeled as algorithms running in polynomial time in* λ *play the following game:*

1. *The challenger runs the key generation algorithm to get a key pair* $(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$ *and publicly announces* pk*.*

[6] the symmetric case is treated similarly.

2. *The adversary can query a decryption oracle $\mathcal{O}^{\mathsf{Dec}_{sk}(.)}$ on arbitrary inputs.*

3. *The adversary arbitrarily chooses two messages of the same size m_0, m_1 and transmits them to the challenger.*

4. *The challenger picks a random bit $b \xleftarrow{r} \{0,1\}$, sets $c \leftarrow \mathsf{Enc}(pk, m_b)$ and returns c to the adversary.*

5. *The adversary can query again the decryption oracle, on any input, except c.*

6. *The adversary outputs a bit b' and wins the game if $b' = b$.*

The encryption scheme PKE *is IND-CCA iff the advantage of the adversary $|Pr[b = b'] - \frac{1}{2}|$ is a negligible function in λ.*

Note that the IND-CCA game we just described is often called the adaptive case or IND-CCA2 game. In the non-adaptive case, the adversary cannot query the oracle anymore after seeing the ciphertext c (the step 5 is skipped).

CPA Security of ElGamal under the decisional Diffie-Hellman assumption

We will now show how to argue the security of an encryption scheme such as ElGamal under a computational assumption. The assumption used here is the hardness of the Decisional Diffie-Hellman problem (DDHP) and is stronger than the one assuming the hardness of the problem described in Definition 4 (computational Diffie-Hellman problem).

Definition 9. *(Decisional Diffie-Hellman problem) Let G be a finite group of order q equipped with a multiplicative law and g one of its generator. Distinguish tuples of the form (g^a, g^b, g^{ab}) for uniformly random a, b in $\mathbb{Z}_q$ from tuples sampled uniformly at random from $G \times G \times G$.*

This captures the fact that keys exchanged during the Diffie-Hellman protocol are indistinguishable from a random group element even for the adversary eavesdropping the values g^a and g^b. If the Decisional Diffie-Hellman assumption holds, an adversary cannot tell the difference between a genuinely generated key g^{ab} and a random value in G.

Using this assumption, a game-based proof can be used to prove semantic security of the ElGamal encryption scheme. Proving security using a sequence of games is a popular technique to prove security properties. The idea is to start from the game modeling the desired property and to iteratively modify it without violating any underlying assumption until reaching a game trivially unwinnable by the adversary. Those transitions define what is commonly called a *sequence of games* and each transition should offer only a negligible advantage to the adversary. A thorough and simple introduction of this technique can be found in [116].

The security of the ElGamal encryption scheme is straightforward using this technique. Let us abstract the adversary as an algorithm $\mathcal{A}$ and define Game 0 as

Game 0

1. $sk \xleftarrow{r} \mathbb{Z}_q, pk \leftarrow g^{sk}$
2. $(m_0, m_1) \xleftarrow{r} \mathcal{A}(pk)$
3. $b \xleftarrow{r} \{0,1\}, r \xleftarrow{r} \mathbb{Z}_q, c_1 \leftarrow g^r, c_2 \leftarrow pk^r \circ \mathsf{H}(m_b)$
4. $b' \xleftarrow{b} \mathcal{A}(pk, c_1, c_2)$

Game 0 is clearly equivalent to the IND-CPA game. The advantage of the adversary is $|Pr[\texttt{WIN}_0] - \frac{1}{2}|$ with $\texttt{WIN}_0$ the event that $b = b'$.

Let us now define a game in which the advantage of the adversary is 0.

Game 1

1. $sk \xleftarrow{r} \mathbb{Z}_q, pk \leftarrow g^{sk}$
2. $(m_0, m_1) \xleftarrow{r} \mathcal{A}(pk)$
3. $b \xleftarrow{r} \{0,1\}, r \xleftarrow{r} \mathbb{Z}_q, c_1 \leftarrow g^r, \underline{e \xleftarrow{r} G}, c_2 \leftarrow \underline{e} \circ \mathsf{H}(m_b)$
4. $b' \xleftarrow{b} \mathcal{A}(pk, c_1, c_2)$

The difference between Game 0 and Game 1 is that instead of using the public key to encrypt $\mathsf{H}(m_b)$ to c_2, we use a random element e from G. It can be seen that the probability $Pr[\texttt{WIN}_1]$ of winning game 1 is actually exactly $\frac{1}{2}$. Indeed, by randomly sampling e, the second part of the ciphertext c_2 is actually an instance of the GOTP offering perfect secrecy. This means that c_2 does not convey any information on m_b. Since c_1 is not linked to the message in any way, the input of the adversary in the last step does not depend on the message at all and the adversary can only guess and win with probability $\frac{1}{2}$. Hence, in this game, the advantage of the adversary is $|Pr[\texttt{WIN}_1] - \frac{1}{2}| = 0$.

Proposition 2. *The ElGamal cryptosystem is semantically secure if the Decisional Diffie-Hellman assumption holds.*

Proof. Let ADV_{DDH} be the probability that an adversary can distinguish randomly drawn triples from G from triples of the form (g^a, g^b, g^{ab}) with $a, b \xleftarrow{r} \mathbb{Z}_q$. The transitional advantage between Game 0 and Game 1 is ADV_{DDH}. Indeed, the difference between the two games is that the value $pk^r = g^{sk \cdot r}$ has been replaced by a random element e from the group. Distinguishing between those two cases is exactly an instance of the Decisional Diffie-Hellman problem which means the advantage of the adversary to distinguish the transition between the two games is also ADV_{DDH}. Hence, the advantage of the adversary against Game 0 is the advantage against Game 1 to which we shall add ADV_{DDH}. Since the advantage of the adversary in Game 1 is 0, their advantage in Game 0 is ADV_{DDH}, which is negligible by hypothesis that the DDHP is hard. □

We note that it can be shown that Proposition 2 is actually an equivalence [120].

2.2.5 From Encryption to Key Exchange: Key Encapsulation Mechanism

A public-key encryption scheme can be easily transformed into a key exchange. Indeed, instead of running a protocol between Alice and Bob, let Alice chose a random key k and encrypt it under the public key of Bob $c = \mathsf{E}(pk, k)$. She then sends c to Bob and he retrieves the shared key k using his secret key. Such a procedure is called a key encapsulation mechanism (KEM).

Definition 10. *A key encapsulation mechanism (KEM) is tuple* $\{\mathcal{K}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps}\}$ *in which:*

- $\mathcal{K}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}$ *are sets called the symmetric keys, ciphertexts, public keys and secret keys spaces.*
- $\mathsf{Gen} : \lambda \rightarrow \mathcal{SK} \times \mathcal{PK}$ *is key generation function taking as input a security parameter and outputting a key pair.*
- $\mathsf{Encaps} : \mathcal{PK} \rightarrow \mathcal{C} \times \mathcal{K}$ *is a function taking as input a public key and outputting a ciphertext together with a symmetric key.*
- $\mathsf{Decaps} : \mathcal{SK} \times \mathcal{C} \rightarrow \mathcal{K}$ *is a function taking as inputs a secret key and a ciphertext and outputting a symmetric key.*

As explained above, creating a KEM from a PKE $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is straightforward. Both Gen algorithms are the same and Decaps is set to Dec. To construct Encaps, one simply has to pick a random k, encrypt it under the public key to get a ciphertext $c = \mathsf{Enc}(pk, k)$ and output c, k.

Semantic security can also be defined for KEM in the following way,

Definition 11. *(IND-CPA game for KEM) Let* $\mathsf{KEM} = (\mathcal{K}, \mathcal{C}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a Key Encapsulation Mechanism and* λ *a security parameter. The challenger and the adversary, both modeled as algorithms running in polynomial time in* λ *play the following game:*

1. *The challenger runs the key generation algorithm to get a key pair* $(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$ *and publicly announces* pk.
2. *The challenger runs* $(c, k_0) \leftarrow \mathsf{Encaps}(pk)$*, picks a random* k_1 *from* $\mathcal{K}$*, a bit* b *and sends* c, k_b *to the adversary.*
3. *The adversary outputs a bit* b' *and wins the game if* $b' = b$.

The key encapsulation mechanism KEM *is IND-CPA iff the advantage of the adversary* $|Pr[b = b'] - \frac{1}{2}|$ *is a negligible function in* λ.

Basically, the mechanism is CPA-secure if the adversary cannot distinguish encapsulated keys from uniformly random keys from the space upon seeing to corresponding ciphertext. Similarly to a PKE, we naturally extend to the CCA-secure version by letting the adversary query a decapsulation oracle $\mathcal{O}^{\mathsf{Decaps}()}$.

2.2.6 Digital Signature

While the goal of encryption was to hide information from an adversary eavesdropping on the communication channel, digital signatures ignore privacy of messages to focus on their genuineness. More precisely, they aim at ensuring that a message actually comes from the claimed source and was not modified along the way by an adversary. The general case is that Alice and Bob are simply communicating and Bob wants to be certain that the messages he reads are coming from Alice and remained untouched during their transmission. Actually, digital signature offer a property quite stronger than that which is called *non-repudiation*. Non-repudiation prevents Alice to claim that she does not agree on a message she previously signed. It means that the signature acts as a "proof of agreement" on a message. Let us say Alice sent a message m stating that she will buy Bob's car for a certain amount of money together with a signature on the message $\sigma(m)$. If she later refuses to proceed with the transaction, Bob can present $\sigma(m)$ to a judge to prove that she committed herself to buy the car.

Digital signature fall into the category of public-key cryptography, which means that entities have key pairs (sk, pk) and are identified by their public keys. The symmetric counterpart of signature is called *Message Authentication Code* and do not offer non-repudiation [7]. Similar to the situation we discussed earlier for encryption, digital signatures, while more powerful, are significantly slower than message authentication codes.

The usual straightforward definition is the following:

Definition 12. *A (digital) signature scheme is a tuple* $(\mathcal{M}, \mathcal{S}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *where:*

- $\mathcal{M}, \mathcal{S}, \mathcal{PK}, \mathcal{SK}$ *are sets called the messages, signatures, public keys and secret keys spaces.*
- $\mathsf{Gen} : \lambda \to \mathcal{SK} \times \mathcal{PK}$ *is key generation function taking as input a security parameter and outputting a key pair.*
- $\mathsf{Sign} : \mathcal{SK} \times \mathcal{M} \to \mathcal{S}$ *is a function taking as inputs a secret key and a message and outputting a signature.*
- $\mathsf{Verify} : \mathcal{PK} \times \mathcal{S} \times \mathcal{M} \to \{0, 1\}$ *is a function taking as inputs a public key and a signature and outputting a bit indicating if the signature is valid for the message.*

Wishing to send a message m to Bob, Alice will first generate a key pair $(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$ using the key generation algorithm. She then runs the signing algorithm using her secret key to get a signature $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and forwards it to Bob together with the message. Upon receiving (m, σ), Bob runs the verification algorithm $\mathsf{Verify}(pk, \sigma, m)$ using her public key and accepts the message as genuine if the verification returns 1.

2.2.7 Basic Security Notions for Signatures

Similarly to encryption schemes, the security of signatures, while pretty intuitive, is formalized using games between a challenger and an adversary. Plethora of variants have been defined and can be found in classical textbooks. For the sake of simplicity, we will

[7] This is because no public information is linked to entities and symmetric keys give the same computational capabilities to every party.

focus on two elementary notions defined in [71] illustrating well the kind security requirement one is looking for. In the following, as usual, the challenger and the adversary are modeled as a polynomial-time algorithms in some implicit security parameter λ.

Existential unforgeability under a random-message attack

Definition 13. *A signature scheme* $(\mathcal{M}, \mathcal{S}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be existentially unforgeable under a random-message attack (UF-RMA) if the probability that an adversary wins the following game is negligible in* λ

1. *A finite, polynomial in* λ*, number of messages* $m_0, \ldots, m_n$ *are chosen at random from* $\mathcal{M}$*.*
2. *The challenger samples a key pair* $(sk, pk) \xleftarrow{r} \mathsf{Gen}(\lambda)$ *and publishes* pk*.*
3. *The challenger computes the signatures* $\sigma_0 \leftarrow \mathsf{Sign}(sk, m_0), \ldots, \sigma_n \leftarrow \mathsf{Sign}(sk, m_n)$ *and send them to the adversary.*
4. *The adversary performs the attack knowing the messages, the signatures and the public key and finishes by outputting a pair* (m, σ)*.*
5. *The adversary wins the game if* $\mathsf{Verify}(pk, \sigma, m) = 1$ *and* $\forall i \in \{0, n\}, m \neq m_i$*.*

This game models the basic scenario in which an adversary cannot control messages that are signed and only sees signatures transmitted on the channel. A weaker notion of security called unforgeability under no-message (UF-NMA) models the case in which the adversary does not even see a single signature before performing the attack.

Unforgeability under a chosen-message attack

Definition 14. *A signature scheme* $(\mathcal{M}, \mathcal{S}, \mathcal{PK}, \mathcal{SK}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be existentially unforgeable under a chosen-message attack (UF-CMA) if the probability that an adversary wins the following game is negligible in* λ

1. *The challenger samples a key pair* $(sk, pk) \xleftarrow{r} \mathsf{Gen}(\lambda)$ *and publishes* pk*.*
2. *The adversary can query the challenger for genuine signatures on any message. The messages for which the adversary made a query are kept in a list* Q*.*
3. *The adversary finishes by outputting a pair* (m, σ)*.*
4. *The adversary wins the game if* $\mathsf{Verify}(pk, \sigma, m) = 1$ *and* $m \notin Q$*.*

In this stronger security model, the adversary can chose himself which message/signature pairs are available to perform the attack. The goal is to model a situation in which someone wishing to compromise a system can somehow convince the owner of the secret key to sign specific messages.

Note that both UF-NMA and UF-CMA security notions described above are said to be the weak versions. In the strong versions, the adversary also wins the game if a new signature on a message *already signed* is found.

Hash functions

Signatures schemes tend to use a cryptographic tool called *hash functions*. Those functions are often well-known as they appear in other areas of computer science even though they do not require the same properties as *cryptographic* hash functions. Since we will use them as black-boxes and not discuss their design nor their security, we simple handwave some general definitions and security properties.

Definition 15. *A hash function is a function* $h : X \to Y, x \mapsto y = h(x)$ *such that* $|X| > |Y|$. *It associate to each element* $x \in X$ *and element* $y \in Y$ *called hash, image, fingerprint or digest of* x.

Definition 16. *A cryptographic hash function is a hash function for which the following problems are intractable:*

- ***One-wayness:*** *Given a value* $y \in Y$*, find a value* $x \in X$ *such that* $h(x) = y$
- ***Second pre-image resistance:*** *Given* $x \in X$*, find a different* $x' \in X$ *such that* $h(x) = h(x')$.
- ***Collision resistance:*** *Find* $x, x' \in X$ *s.t.* $h(x) = h(x')$

Note that since the input space is bigger than the output space, there always exist collisions. An additional common property is called the *avalanche property*. Informally, it states that a small perturbation in the inputs should result in a huge variation in the output. Generally, cryptographic hash functions are defined as a mapping from $\{0,1\}^*$ to $\{0,1\}^n$ but there also exists extendable-output functions mapping to arbitrary size outputs. Those are regularly used in cryptographic schemes to generate pseudo-random numbers from a seed or when a fixed-length digest is required by the scheme.

2.2.8 Identification Schemes

A well-known, and very relevant to this thesis, way of creating signature schemes is to start from an identification scheme and to transform it using a technique called the Fiat-Shamir transform. This technique will be explained in Section 2.2.9 but we shall first of course briefly discuss the underlying topic which is identification schemes.

Identification schemes are protocols between two parties: a prover P and a verifier V. The goal is that at the end of the protocol, the verifier is convinced that prover is indeed P and not an impostor. It is easy to imagine situations in which this is useful (e.g. opening a locked door) as it is a basic requirement of access control.

One simple way to create an identification scheme is to use a hash function h (or any one-way function) and set the identity of P to $y = h(x)$ for some secret x chosen by P. When P wants to prove its identity to V, the value x is transmitted and V checks whether y is actually equal to $h(x)$. If it is the case, it proves the identity since P is the only person capable of providing a preimage of y. This is basically the scenario of the login/password authentication system used everywhere on the internet but it is very weak in a vacuum since it reveals the secret x to the verifier and anyone eavesdropping on the channel[8]. Thus, we will be interested in a stronger version using a public-key setting called canonical identification schemes.

[8]When login to a website, the connection is (normally) already encrypted and the server is assumed to be trusted.

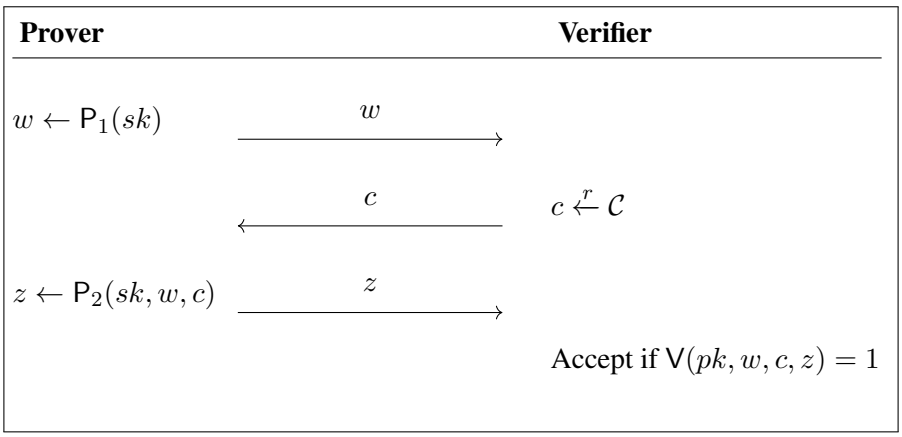

Figure 2.3: Canonical identification scheme

Definition 17. *A canonical identification scheme is a tuple* $(\mathcal{PK}, \mathcal{SK}, \mathcal{W}, \mathcal{C}, \mathcal{Z}, \mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *in which:*

- $\mathcal{PK}, \mathcal{SK}, \mathcal{W}, \mathcal{C}$ *and* $\mathcal{Z}$ *are called the public keys, secret keys, commitments, challenges and responses spaces.*
- $\mathsf{Gen} : \lambda \rightarrow \mathcal{SK} \times \mathcal{PK}$ *is key generation function taking as input a security parameter and outputting a key pair.*
- $\mathsf{P} = \{\mathsf{P}_1, \mathsf{P}_2\}$ *is a set of functions defining the prover side such that* $\mathsf{P}_1 : \mathcal{SK} \rightarrow \mathcal{W}$ *and* $\mathsf{P}_2 : \mathcal{SK} \times \mathcal{W} \times \mathcal{C} \rightarrow \mathcal{Z}$
- $\mathsf{V} : \mathcal{PK} \times \mathcal{W} \times \mathcal{C} \times \mathcal{Z} \rightarrow \{0, 1\}$ *is the verification function*

The identification procedure is a three-steps protocol between the prover and the verifier. Let us assume the prover generated a key pair (sk, pk). It starts with the prover running P_1 to get a value w called the commitment that is sent on the channel. The verifier will then reply with a randomly chosen $c \xleftarrow{r} \mathcal{C}$ from the challenge space. Using the previously chosen commitment w and the newly received challenge c, the prover computes a response $z \leftarrow \mathsf{P}_2(sk, w, c)$. Finally, the verifier accepts if $\mathsf{V}(pk, w, c, z) = 1$. If the tuple (w, c, z) (we let the public key be implicit) is such that the verification is a success, we call (w, c, z) an *accepting transcript*. Figure 2.3 illustrates a canonical identification scheme. Such three-moves protocols are sometimes called Σ-protocols.

Before moving on to the Fiat-Shamir transform, let us define two common properties facilitating security arguments for identification schemes.

Definition 18. *(Honest-verifier zero-knowledge) An identification scheme has the honest-verifier zero-knowledge (HVZK) property if there exists an efficient simulation algorithm* sim *taking as input only a public key and outputting an accepting transcript indistinguishable from genuine executions of the protocol between a prover holding the corresponding private key and an honest verifier*[9].

This properties aims at providing security against passive adversaries. Indeed, if anyone can create accepting transcripts at will using only public values, it means that no

[9]By honest, we mean a verifier simply following the protocol as described above.

information on the secret is gained through the observation of accepting transcripts. In particular, observation of genuine transcript created by an execution of the protocol between the holder of the secret key and an honest verifier will not reveal information on the said key.

Definition 19. *An identification scheme has the special soundness property if it is computationally infeasible for an entity not knowing the secret key to find two accepting transcripts* (w, c, z) *and* (w', c', z') *such that* $w = w'$ *and* $c \neq c'$.

The intuition behind this property is that if it is impossible to find two transcripts (w, c, z) and (w', c', z') such that $w = w'$ and $c \neq c'$, any adversary able to attack the identification scheme without recovering the secret key has a probability of success at most $\frac{1}{|\mathcal{W}|}$, which can be made negligible by picking a large enough set $\mathcal{W}$. Indeed, say an adversary attacking the scheme with commitment w got a challenge c and output a response z such that (w, c, z) is a valid transcript and that it would have been impossible to attack on (w, c'). Since the adversary cannot know a priori which challenge will be sent by V and, thus, the commitment w was chosen independently of c, it would have been impossible that the attack succeeds on any $c' \neq c$, which would mean that the probability of success is upper bounded by $Pr[c \xleftarrow{r} \mathcal{W}] = \frac{1}{|\mathcal{W}|}$.

2.2.9 From Identification to Signatures: The Fiat-Shamir Transform

The reason why we introduced identification scheme (beside the fact that it is an interesting topic!) is that there exists a way to construct a signature from a canonical identification scheme called the *Fiat-Shamir transform*.

The idea is for a prover to play the authentication protocol alone while making the game depend on the message. Hence, the transcript of the game correspond to a signature on the message and anyone in possession of the public key can play the role of the verifier later on to verify the validity of the signature. One should be careful here because since identification games having the HVZK property are simulable, a signature should correspond to a genuine execution of the protocol. The solution is to force the prover to simulate a verifier using what is called a random oracle.

Random Oracle

Definition 20. *A random oracle is a black box taking as input arbitrary values and returning uniformly random values from its output space but that stays consistent when queried multiple times on the same input.*

Random oracle (RO) are theoretical objects that cannot be instanciated in practice but can easily be simulated for a polynomial number of queries. Indeed, the simulator will store a list of pairs of the form (x_i, y_i) in which x_i is the $i\text{-}th$ query and y_i the $i\text{-}th$ answer. Upon receiving a new query x, the simulator performs a search in the list and outputs y_i if $x = x_i$ for some i. If not, a value y chosen uniformly at random from the output set is returned and the pair (x, y) added to the list. Since random oracles are not instanciable in the real world, they are approximated with classical hash functions. While this functions are deterministic and, thus, not random oracles, from the point of view of a user, a "good" hash function is indistinguishable from a random oracle as it is obviously consistent but will return a random looking output on a new fresh input due to the one-wayness property concealing the relation between outputs and inputs and the avalanche property concealing the link between close inputs.

Let $(\mathsf{Gen}, \mathsf{P} = \{\mathsf{P}_1, \mathsf{P}_2\}, \mathsf{V})$ be a canonical identification scheme with associated usual spaces and $H : \{0,1\}^* \to \mathcal{C}$ a random oracle. The key generation algorithm of the signature scheme is simply Gen.

Fiat-Shamir Sign(sk, m):

1: $w \leftarrow \mathsf{P}_1(sk)$
2: $c \leftarrow H(w, m)$
3: $z \leftarrow \mathsf{P}_2(sk, w, c)$
4: **return** $\sigma = (w, z)$

Fiat-Shamir Verify$(pk, \sigma = (w, z), m)$:

1: $c \leftarrow H(w, m)$
2: **return** $\mathsf{V}(pk, w, c, z)$

Figure 2.4: Fiat-Shamir signature based on a canonical identification scheme

The Fiat-Shamir transform

As explained above, the goal of the Fiat-Shamir transform is to modify an identification scheme such that it depends on the message and corresponds to a genuine execution with an honest verifier. Both those constraints are met by using a random oracle H having the challenge space as output space and simulating a verifier by computing the challenge as $c \leftarrow H(w, m)$ with m the message to sign and w the commitment. Indeed, in this case, the game obviously depends on the message and flows as a genuine execution because

1. the challenge is uniformly distributed and,
2. the temporality of the protocol is maintained since the challenge cannot be computed before w is chosen.

The Fiat-Shamir transform is more formally described in Figure 2.4. Note that for identification schemes in which w is easy to compute from pk, c and z, the signature output is often (c, z) instead of (w, z) because it generally has a shorter representation.

Schnorr signature

Figure 2.5 presents a signature scheme constructed using the Fiat-Shamir transformation called Schnorr signature [111]. The scheme base its security on the hardness of the discrete logarithm problem and is a simple illustrative example of a Fiat-Shamir signature.

Showing unforgeability properties of this signature involves arguments that are out of scope of this introduction and will not be done here. Nevertheless, we would like to point out three facts:

- Under the discrete logarithm assumption, y is hard to compute from w and thus unknown to an eavesdropper. This means that z is independent of the secret key sk (and thus does not reveal any information on it) under the GOTP perfect secrecy property (Definition 6).
- Under the discrete logarithm assumption, the underlying identification scheme achieves special soundness. Indeed, for the public key $pk = g^{sk}$, from two transcripts (w, c, z) and (w', c', z') with $w = w'$ and $c \neq c'$, the adversary can easily extract $sk = (z - z') \cdot (c - c')^{-1}$ and, thus, solve the discrete logarithm problem.

Public parameters:

- (multiplicative) cyclic group G of order q generated by g
- random oracle $H : \{0,1\}^* \to \mathbb{Z}_q$

Signing key: $sk \xleftarrow{r} \mathbb{Z}_q$
Verifying key: $pk \leftarrow g^{sk}$

Schnorr Sign(sk, m):

1: $y \xleftarrow{r} \mathbb{Z}_q$
2: $w \leftarrow g^y$
3: $c \leftarrow H(w, m)$
4: $z \leftarrow y + sk \cdot c \mod q$
5: **return** $\sigma = (c, z)$

Schnorr Verify$(pk, \sigma = (c, r), m)$:

1: $w' \leftarrow g^z \cdot pk^{-c}$
2: **return** $c \stackrel{?}{=} H(w', m)$

Figure 2.5: Schnorr signature

- The underlying protocol has the HVZK property. Picking c and z at random in $\mathbb{Z}_q$ and setting $w \leftarrow g^z \cdot pk^{-c}$ gives accepting transcripts (w, c, z) that can be shown to perfectly simulate a genuine execution with an honest verifier.

The Schnorr signature, beside its simplicity, is of great interest for this thesis as it is also the basis on which popular lattice-based signatures scheme from Chapter 4 were built.

2.3 Post-Quantum Cryptography

As explained in the informal high level introduction to cryptography, modeling and assessing the capabilities of the adversary is a fundamental requirement to build useful cryptographic schemes. Indeed, if a scheme is proven secure in a model that does not correspond to any real-life scenario, the security proof is worthless.

The most common assumption generally made is that we face a probabilistic polynomial-time (PPT) adversary. That is to say, an adversary that uses algorithms running in polynomial time in some security parameter and that can access some randomness during its computation. One implicit hypothesis is that the adversary uses a *classical* computer. In a world in which the adversary has access to a *quantum* computer, some security assumptions, especially on the hardness of several cryptographic problems, do not hold anymore. Adding quantum computers to the list of possible tools, one can see three cryptographic scenarios:

1. **Classical Cryptography**: Everyone, adversaries and legitimate participants, is using a classical computer. This is the scenario that as been mostly considered through the history of cryptography.
2. **Post-Quantum Cryptography**: The adversaries are equipped with quantum computers while the honest participants have to defend using solely classical computations. This is the topic of this thesis.

3. **Quantum Cryptography**: Everyone, adversaries and legitimate participants, is using a quantum computer. While this scenario could somehow make sense in a distant future, it assumes that every device using cryptography is quantum capable and that there always exists a quantum communication channel, which is quite unrealistic as of now.

The scenario in which the adversary is classical and the honest participants are quantum is irrelevant since it is basically the same (or better) as classical cryptography from the point of view of the defender. In this thesis, we care about the second scenario, the one that models a world in which a powerful adversary, for example a large company or a state, get access to a quantum computer and tries to use it to attack some cryptographically secured communication. Before stating why it is important to care about this scenario right now, we should first explain what is alarming about it. While no full-fledged large scale quantum computer has been publicly built yet, a large effort studying its algorithmic capabilities from an abstract model have been made during the past decades. The scariest result concerning cryptography is Shor's algorithm. In 1994, Peter Shor described a quantum algorithm [115] that can solve both the integer factorization and the discrete logarithm problem in polynomial time. This means that, in a quantum world, all the cryptographic schemes described earlier which base their security on the Diffie-Hellman problem (and thus discrete logarithm) are insecure. Actually, the bad news is that almost all public-key schemes used in practice right now are based either on the discrete logarithm (for specific groups)or the integer factorization problems. For example, the TLS protocol, which is used all over the Internet, is currently only using key exchanges and signature schemes insecure against a quantum adversary. Explaining the details of Shor's algorithm require a decent background in quantum computing and is not required to understand the practical challenges post-quantum cryptography is facing. Thus, in this thesis, we will mostly ignore how a quantum computer is working and focus solely on schemes that are believed to resist quantum attacks. The curious reader is redirected to the excellent textbook of Nielsen and Chuang [96]. The reason why this method is sound is that, for practical purposes, as long as the scheme is proven to be secure under a quantum-resistant problem, the fact that the adversary is quantum is irrelevant in theory [10]. Actually, post-quantum schemes were not really crafted to counter a quantum adversary, what often happened is that an already existing scheme that had not been extensively studied (because it offered less efficiency than discrete logarithm/factorization-based algorithms) gained some popularity because no quantum attacks against it are known. For example, the McEliece cryptosystem [87] was described in 1978, almost twenty years before the groundbreaking work of Shor, and its modern derivatives are considered as viable post-quantum candidates. This phenomenon is somewhat due to fundamental open questions in complexity theory. Since the **P** vs **NP** problem is still unsolved, we do not even know if hard problems in the cryptographic sense do exist. Hence, instead of creating hard problems based on the adversary, we instead pick existing problems that are strongly believed to be out of reach. Since the same kind of questions are open for quantum computing, post-quantum cryptography also relies on problems only believed to be hard for a quantum computer. Actually, it is not even known whether the classes of efficiently solvable problems on a classical (**P**) and on a quantum (**BQP**) computer are different. Since quantum resistant cryptographic algorithms can be built from several hard problems, they are regrouped in different categories. The main ones are:

[10] In practice, one should use stronger parameters sets against quantum adversaries.

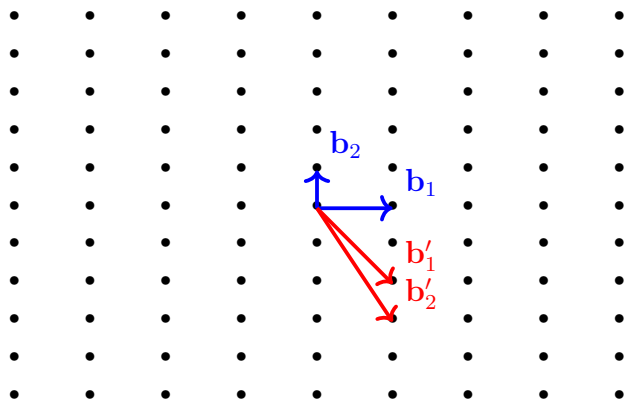

Figure 2.6: Lattice together with two of its basis, $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}\}$ and $\mathbf{B}' = \{\mathbf{b}'_1, \mathbf{b}'_2\}$

- Lattice-based cryptography;
- Code-based cryptography;
- Isogeny-based cryptography;
- Hash-based cryptography;
- Multivariate cryptography.

Each category has its advantages and drawbacks and choosing among them for standardization is not an easy task. This thesis focus solely on lattice-based cryptography but the goal is not to advocate for the superiority of this category. The personal view of the author is that lattice-based cryptography is promising because of its versatility, plethora of cryptographic constructions can be built based on lattice problems and it gives a unified framework to study them. Beside, their structured variants seem to offer relatively good trade-offs between bandwidth and efficiency. Nevertheless, this kind of considerations are always context dependent. Sometimes having large keys is not an issue, sometimes slow computations are acceptable, there is not a clear undebatable metric here.

2.3.1 Lattices

We now move on to the main tools used to construct the cryptographic schemes that are studied in this thesis. This shallow introduction to lattices aims to provide to the reader a high level overview of the field and useful definitions required to understand upcoming chapters. The theoretical work behind lattices, both on the mathematical and cryptographic sides is quite deep an clearly out of scope. More details and references about the history of lattice cryptography can be found in the excellent survey of Peikert [101] that has been very useful to write this section.

A n dimensional lattice Λ is an additive discrete subgroup of $\mathbb{R}^n$. It is composed of the set of vectors that can be expressed as a linear combination of linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}$ with $\mathbf{b}_i \in \mathbb{R}^n$, $m \leq n$. More precisely:

$$\Lambda(\mathbf{B}) = \left\{ \sum_{i=1}^{m} a_i \cdot \mathbf{b_i} | a_i \in \mathbb{Z} \right\} \tag{2.1}$$

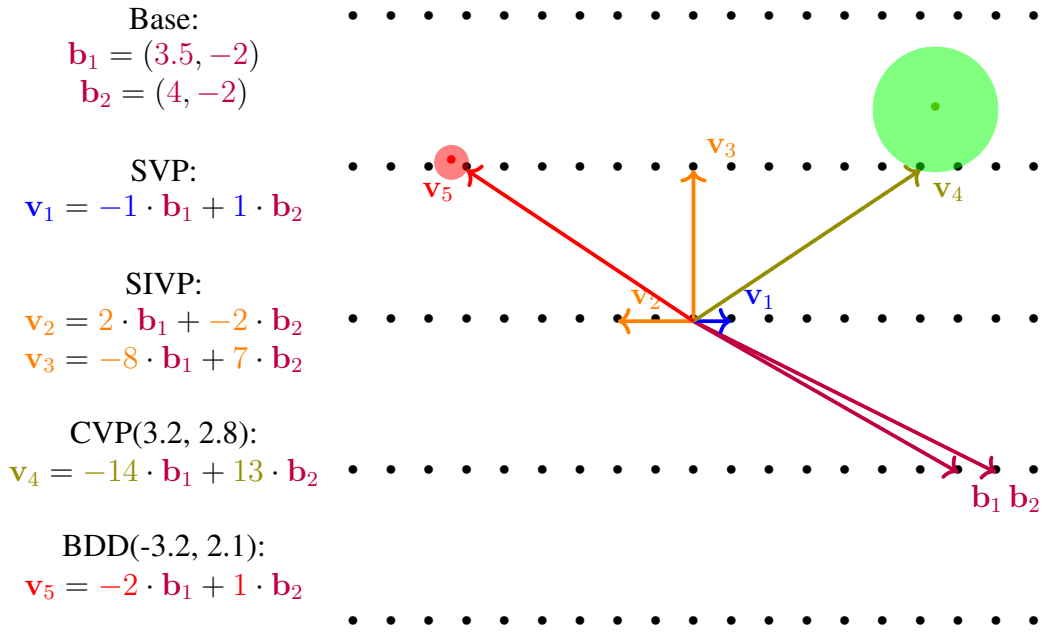

Figure 2.7: All-in-one figure showing the different basic lattice problems together with their solutions in a simple lattice of dimension 2.

The matrix $\mathbf{B} \in \mathbb{R}^{n \times m}$ is called *the basis* of the lattice λ. If $m = n$, the lattice is said to be full-rank. This will be assumed.

A lattice can be expressed by an infinite amount of basis. For all unitary matrix $\mathbf{U} \in \mathbb{Z}^{n \times n}$, $\Lambda(\mathbf{B}) = \Lambda(\mathbf{B} \cdot \mathbf{U})$. When the basis is clear from context, we simply write $\mathbf{B}$.

Definition 21. *(minimum distance) The minimum distance of a lattice Λ, denoted $\lambda_1(\Lambda)$ is the quantity* $\min_{\mathbf{v} \in \Lambda, \mathbf{v} \neq 0} \|\mathbf{v}\|$.

Said otherwise, it is the norm of a shortest nonzero vector of the lattice. Furthermore, we can also define λ_2 as the second shortest vector not on the same line as λ_1, then, λ_3 as the shortest vector not in the span of $\{\lambda_1, \lambda_2\}$ and so on and so forth.

Definition 22. *(successive minimum distances) The i-th successive minimum distance λ_i is defined as the minimum norm of the largest element in a set of i linearly independent lattice vectors. More formally, it is defined as $\lambda_i = \min_r \{r \mid dim(span(\mathcal{B}(0, r) \cap \Lambda)) \geq i\}$ with $dim(V)$ the dimension of the vector space V, $span(S)$ the vector subspace spanned by S and $\mathcal{B}(\mathbf{c}, r)$ the ball of radius r centered in $\mathbf{c}$.*

2.3.2 Hard Problems on Lattices

Lattices are simple to construct geometric objects but nevertheless offer some easy to define hard problems. The algorithmic challenges offered by lattices appeared in computer sciences prior to their usage in cryptography and, even before, the theory of lattices has been studied in the mathematical literature in fields such as *the geometry of numbers*. This extensive scrutiny of lattice related problems grants a certain level of confidence toward their hardness, which is very valuable in cryptography. However, it should be mentioned that concrete instanciations are pretty far from the original lattice problems as they are only equivalent through non-tight reductions. The problems presented below will be referred to as the *basic lattice problems*.

Shortest vector problem

The first problem we define asks to find short vectors in the lattice from an arbitrary basis. It is called the Shortest Vector Problem (SVP).

Definition 23. *(SVP) Given a basis* $\mathbf{B}$ *of a lattice* Λ*, find a vector* $\mathbf{v}$ *such that* $\|\mathbf{v}\| = \lambda_1(\Lambda)$.

Definition 24. *(SVP$_\gamma$) Given a basis* $\mathbf{B}$ *of a lattice* Λ *of dimension* n *and an approximation factor* $\gamma = f(n)$*, find a vector* $\mathbf{v}$ *such that* $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\Lambda)$.

The function f plays an important role in the hardness of this problem. If we take $f(n) = 1$, it is equivalent to the basic SVP problem, which is NP-hard. For f polynomial in n, we lose the NP-Hardness in general but the problem is still believed to be intractable, even for a quantum adversary. For exponential f, polynomial time algorithms solving those problems exist, the most famous being the basis reduction algorithm LLL [76].

Similar to the discrete logarithm problem, there also exists a decisional version.

Definition 25. *(gapSVP$_\gamma$) Given a basis* $\mathbf{B}$ *of a lattice* Λ *of dimension* n *and a value* $\gamma = f(n)$*, decide if* $\lambda_1(\Lambda) \leq 1$ *or if* $\lambda_1(\Lambda) > \gamma$*. If* $\lambda_1(\Lambda)$ *is in between, the decider algorithm can output an undefined result.*

Shortest Independent Vector Problem

The Shortest Independent Vector Problem (SIVP) is a sibling of SVP in which the goal is not to find a shortest vector but a set of linearly independent vectors such that the norm of the largest vector in the set is minimal.

Definition 26. *(SIVP) Given a basis* $\mathbf{B}$ *of a lattice* Λ *of dimension* n*, find a set of linearly independent vectors* $\mathbf{V} = \{\mathbf{v}_i\}$ *such that* $\max_i \|\mathbf{v}_i\| = \lambda_n(\Lambda)$.

Definition 27. *(SIVP$_\gamma$) Given a basis* $\mathbf{B}$ *of a lattice* Λ *of dimension* n *and an approximation factor* $\gamma = f(n)$*, find a set of linearly independent vectors* $\mathbf{V} = \{\mathbf{v}_i\}$ *such that* $\max_i \|\mathbf{v}_i\| = \gamma \cdot \lambda_n(\Lambda)$.

Closest Vector Problem

The last lattice problem we define is called the Closest Vector Problem (CVP). In this one, we work with points that are in the ambient space and try to determine where is the closest lattice point.

Definition 28. *(CVP) Given a basis* $\mathbf{B}$ *of a lattice* Λ *and a target point* $\mathbf{t} \in \mathbb{R}^n$*, find the vector* $\mathbf{v} \in \Lambda$ *such that the distance between* $\mathbf{t}$ *and* $\mathbf{v}$ *is minimal.*

The Closest Vector Problem also offers a variant in which a lattice point close to the target is ensured to exist, it is called the Bounded Distance Decoding (BDD) problem.

Definition 29. *(BDD) Given a basis* $\mathbf{B}$ *of a lattice* Λ *and a target point* $\mathbf{t} \in \mathbb{R}^n$*, find the vector* $\mathbf{v} \in \Lambda$ *closest to* $\mathbf{t}$*, given that there exists* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{t} - \mathbf{v}\| < \lambda_1(\Lambda)/2$.

Figure 2.7 shows a graphical representation of the different problems briefly discussed in this section.

2.3.3 SIS, LWE and their Algebraically Structured Variants

Having difficult problems is a natural requirement to construct cryptography. Nevertheless, these problems do not tell how to construct efficient schemes from them. In lattice-based cryptography, it is often the case that the proposed cryptosystem is actually based on an intermediate problem that has a reduction from the basic lattice problems described in the previous section. Among them, the two problems relevant to this thesis are the *Short Integer Solution* (SIS) and the *Learning With Errors* (LWE) problems. For completeness, we must also cite the famous NTRU cryptosystem [21, 65, 123] that belongs to the lattice family but is based on related but different problems that the ones we will develop now.

Short Integer solution

One of the first fundamental lattice related problem is called the *Shortest Integer Solution* (SIS) problem. It has been first described by Ajtai [1] in a paper seen as the cornerstone of a large amount of lattice-based cryptosystem.

Definition 30. *(SIS) Given a random matrix* $\mathbf{A} \in \mathbb{Z}^{n \times m}$ *and a bound* b*, find a vector* $\mathbf{s} \in \mathbb{Z}^m$ *such that* $\|\mathbf{s}\| < b$ *and* $\mathbf{A} \cdot \mathbf{s} \equiv \mathbf{0} \mod q$.

Basically, the problem is to find a small linear combination of the columns of a matrix in $\mathbb{Z}_q^{n \times m}$ giving the zero vector. This shares similarities with the famous knapsack problem. The bound b is crucial to its hardness because otherwise, simple Gaussian elimination yield a valid solution. The problem is obviously meaningful if there exists at least a solution, this can be forced by choosing relevant parameters. For example, if we take $m > n \log_2 q$ and $b > \sqrt{m}$, we have more vectors in $\{0,1\}^m$ than in $\mathbb{Z}_q^n$, hence there exists $\mathbf{v}$ and $\mathbf{v}'$ such that $\mathbf{A} \cdot \mathbf{v} = \mathbf{A} \cdot \mathbf{v}'$. The vector $\mathbf{s} = \mathbf{v} - \mathbf{v}'$ is such that $\mathbf{A} \cdot \mathbf{s} \equiv \mathbf{0}$ and $\|\mathbf{s}\| < b$.

The SIS problem can be seen as an average-case (approximate) SVP problem over a specific class of lattices. Indeed, let us consider the lattice

$$\Lambda(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^m \mid \mathbf{A} \cdot \mathbf{v} \equiv \mathbf{0} \in \mathbb{Z}_q^n, \}$$

a short vector in such a lattice is precisely a solution to the SIS problem. Nevertheless, while this result is connecting the SIS problem and lattices problems from Section 2.3.2, it does not tell much about the hardness of the problem since it might be the case that finding short vectors in such lattice is actually much easier than the general case. Fortunately, way stronger results for this problem are known. When Ajtai introduced SIS, his paper came with a *worst-case to average-case reduction* (that has been studied further by other researchers in the literature) from several basic lattice problems such as gapSVP$_\gamma$ and SIVP$_\gamma$ with $\gamma = \tilde{\mathcal{O}}(n)$ [90], to the SIS problem. This means that an adversary capable to efficiently solve a randomly chosen instance of SIS with non-negligible probability can be transformed in polynomial time into an adversary solving some basic lattice problem on *an arbitrary* lattice! This result is actually very strong because it indicates that the average instance of the SIS problem is theoretically as hard as *the hardest* instance of the base problem. Nevertheless, it should be mentioned that the concrete cryptosystems based on SIS *are not* instanciated according to this reduction since it would make them impractical due to the non-tightness of the proof. This result should be more realistically seen as an indication that the problem is hard by nature.

A simple collision resistant function from SIS

As a proof of concept of cryptographic primitive based on SIS, we show a simple construction of a collision resistant function. Let $\mathbf{A} \in \mathbb{Z}^{n \times m}$ be a matrix such that the SIS problem with $b > \sqrt{m}$ is hard, the function

$$f : \{0,1\}^m \to \mathbb{Z}_q^n : \mathbf{x} \mapsto \mathbf{A} \cdot \mathbf{x}$$

is collision resistant. Indeed, finding two inputs $\mathbf{x}$ and $\mathbf{x}'$ such that $f(\mathbf{x}) = f(\mathbf{x}')$ trivially gives a vector $\mathbf{s} = \mathbf{x} - \mathbf{x}'$ which is a solution to the SIS problem.

Learning With Errors

The flagship problem of modern lattice-based cryptography is called *Learning With Errors* (LWE). It offers myriad of applications and was introduced by Regev [109] in 2005 . The author subsequently won the prestigious Gödel prize in 2018 for his analysis of the problem. It can be used for very efficient constructions of public-key encryption and signatures but also offers the possibility to create advanced primitives such as fully homomorphic encryption. Also enjoying valuable hardness results, its popularity grew strongly with the urgent need for post-quantum primitives.

Let q and n be positive integers and χ a zero mean narrow distribution over $\mathbb{Z}$ which is usually a discrete Gaussian or a binomial distribution.

Definition 31. *(search-LWE) For a fixed* $\mathbf{s} \in \mathbb{Z}_q^n$*, given* m *samples* $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \mod q)$ *with the* $\mathbf{a}_i$ *sampled uniformly from* $\mathbb{Z}_q^n$ *and the* e_i *sampled from* χ*, find* $\mathbf{s}$*.*

This search version of LWE is more commonly stated as solving a noisy system of equations over $\mathbb{Z}_q$: given a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and a target vector $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \mod q$ with coefficients of the m dimensional vector $\mathbf{e}$ sampled from χ, find $\mathbf{s}$.

Similarly to SIS, the search-LWE problem can be expressed as a basic lattice problem. Let us consider the lattice

$$\Lambda(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^m \mid \mathbf{A} \cdot \mathbf{s} \equiv \mathbf{v} \mod q\},$$

search-LWE is equivalent to a BDD problem.

An even more powerful version of this problem, which renders the LWE problem somewhat analogous to the Decisional Diffie-Hellman problem is the following version:

Definition 32. *(decision-LWE) Given* m *samples* $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ *with each* $\mathbf{a}_i$ *sampled uniformly at random, decide if the* b_i *were also sampled uniformly at random or if they were computed as* $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \mod q$ *for a fixed* $\mathbf{s}$ *and each* e_i *sampled from* χ*.*

Basically, this problem asks to distinguish LWE samples from uniformly random elements from $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Similarly to the decisional Diffie-Hellman problem, this is really useful to argue the semantic security of encryption schemes. This problem is also more commonly described by regrouping the samples in a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$.

Equivalently to SIS, the hardness of the learning with errors problem is based on a worst-case to average case reduction. In his original work, Regev provided a quantum reduction from gapSVP$_\gamma$ and SIVP$_\gamma$ on arbitrary lattices to decision-LWE for $\gamma = \tilde{\mathcal{O}}(n/\alpha)$ where $0 < \alpha < 1$ is a parameter related to the standard deviation αq of the error distribution.

The reduction being quantum means that an adversary who can solve LWE and who have access to a quantum computer can also solve gapSVP$_\gamma$ and SIVP$_\gamma$. The results has been improved later with the apparition of fully classical worst-case to average-case reductions for LWE [32, 99]. It has also been showed that picking the secret $\mathbf{s}$ from the *same distribution as the error* does not affect the hardness of the problem (and others LWE related problems). While it is not a tremendous change to the definition, it is a really useful property in practice, crucial to the construction of certain schemes.
We also note that the problem stays hard for different instances sharing the same $\mathbf{a}_i$ but with different secrets. For instance, it is still hard to distinguish tuples of the form $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s}_1 \rangle + e_{1,i}, \langle \mathbf{a}_i, \mathbf{s}_2 \rangle + e_{2,i})$ from tuples sampled uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_q^2$. Indeed, with such a distinguisher, upon receiving a pair of the form $(\mathbf{a}_i, b_i)$, one could craft a new sample $b_i' = \langle \mathbf{a}_i, \mathbf{s}' \rangle + e_i'$ and call the distinguisher on $(\mathbf{a}_i, b_i, b_i')$ to solve decision-LWE. Intuitively, since the $\mathbf{a}_i$ are public, new samples with different secrets can be computed by anyone and thus, do not help distinguishing.

Ring-LWE

It is natural that the number one characteristic of a cryptograhic problem should be security. Nevertheless, to be useful in the real world, outside of academic papers, it must also be efficient. Depending on the context, this efficiency can be expressed in terms of different metrics but the most common ones are speed, key sizes or ciphertext/signature size. Lattices constructions based on SIS or LWE are known to suffer greatly from large key sizes and this issue spanned a line of research studying some algebraically structured variants offering better performances at the price of more specific assumptions. Studying all the variants and there subtleties requires a quite strong arsenal of tools from algebraic number theory and would be well out of scope of this thesis. We redirect the interested reader to [102] and the numerous references therein for an overview of the different flavors. For our purpose, we restrict ourselves to the most commonly used variant. With q and χ defined as in the plain LWE section, let $\mathcal{R}_q$ be the ring $\mathbb{Z}_q[X]/\langle \phi_n \rangle$ with ϕ_n the n-th cyclotomic polynomial [11] and where n is often chosen as a power of two (which means ϕ_n is of the form $X^{2^k} + 1$ and $n = 2^{k+1}$).

Definition 33. *(search-RLWE) For a fixed $s \in \mathcal{R}_q$, given m samples $(a_i, b_i = a_i \cdot s + e_i)$ with the a_i sampled uniformly from the ring $\mathcal{R}_q$ and the $e_i \in \mathcal{R}_q$ having coefficients sampled from χ, find s.*

At first sight, it looks like not much has been gained from the original LWE problem, the inner product over $\mathbb{Z}_q^n$ has been replaced by the product in the n dimensional ring $\mathcal{R}_q$. In fact, the huge performance gain comes from the fact that the b_i are n dimensional in RLWE while they are only a scalar in LWE. This means that the number of sample needed for a given cryptographic application can be divided by n. Actually, most scheme based on RLWE only use a couple of samples while some LWE schemes can use up to thousands of samples. Also, on the computational side, thanks to fast polynomial muliplication algorithms, schemes based on RLWE are faster than their LWE counterpart using the same dimension. Naturally, RLWE also offers a decisional variant.

Definition 34. *(decision-RLWE) Given m samples $(a_i, b_i) \in \mathcal{R}_q \times \mathcal{R}_q$ with each a_i sampled uniformly at random, decide if the b_i were also sampled uniformly at random or if*

[11] The n-th cyclotomic polynomial is the product $\prod_{i \in \{1,\dots,\varphi(n)\}} (x - \omega_i)$ with ω_i the primitive roots of unity of order n.

they were computed as $b_i = a_i \cdot s + e_i$ *for a fixed* $s \in \mathcal{R}_q$ *and each* $e_i \in \mathcal{R}_q$ *having coefficients sampled from* χ.

On the side of hardness, results similar to the non-structured case are known. The difference is that reductions are defined for basic lattice problems over a special class of lattices called *ideal lattices*. The crucial question is naturally how both problems compare to each other in terms of security. Of course RLWE cannot be harder than LWE but its structure does not seem enable any devastating attack. More details in [83, 84]. Actually, this idea of giving algebraic structure to the problem was first applied to SIS [89].

Definition 35. *(R-SIS) Let* $\mathcal{R}$ *be a ring,* b *an integer and* $\mathbf{a} \in \mathcal{R}^m$*, find a nonzero* $\mathbf{s} \in \mathcal{R}^m$ *such that* $\langle \mathbf{a}, \mathbf{s} \rangle = \sum_i \mathbf{a}_i \cdot \mathbf{s}_i = \mathbf{0}$ *and* $\|\mathbf{s}\| < b$.

The results for this problem regarding efficiency of constructions and hardness are similar to the ones of RLWE. Therefore, we will not expand more on this topic.

2.3.4 NIST Standardization Process

A question we ignored until now is the usefulness, or more precisely, the need for post-quantum algorithms. Surely, advances in quantum computing are made here and there, but nothing concrete that would weaken cryptography has been announced yet. The threat of quantum cryptanalysis is currently only speculation from a theoretical model. Even though proof of concept implementations on small quantum computers have been demonstrated, maybe some unknown physical constraint actually undermines the realization of a large scale quantum machine. Yet, the threat seems to be real and growing and even if some are still reluctant to accept the necessity of post-quantum cryptography, a large amount of people think that the risks are too high to be ignored. Hence, the field of post-quantum cryptography has been growing a lot in the past years. The reason why the research community is getting so active a long time prior to the creation of the first large quantum computer is because a lot of things take time before having a full-fledged ready to deploy solution. Mainly,

- Building secure and practical schemes;
- Getting confidence in the underlying assumption;
- Deploying the scheme outside of academia.

And beside that, a really common argument in favor of putting more forces toward quantum-resistant schemes is that some messages might need to keep their security for an extended amount of time. If some information has to stay secret for, say, twenty years, the encryption method used to encrypt it has to stay unbreakable for twenty years. This means that post-quantum security might already be needed right now.

In this context, at the dawn of 2017, the National Institute of Standards and Technology (NIST) started the Post-Quantum Cryptography Standardization project. The goal was to create a framework in which researchers could officially propose concrete solutions for encryption (or key encapsulation) and signatures in a post-quantum world. The original deadline for proposal was at the end of November 2017. After this date, all the proposed schemes, 69 candidates, became public and the "competition" entered its first round. Since all candidates had to provide a detailed specification document together with concrete parameter sets and at least an implementation, this motivated a lot of research.

Attacks on some proposals rapidly flourished and improvements were made. In early January 2019, the project started its second round. While the first one was more about the design of the schemes, NIST announced that performances will have more importance in this second phase. Among the original candidates, 17 encryption/key encapsulation schemes and 9 signature schemes were selected to advance. At the time of writing this section, the second round is still ongoing. The lattices candidates in the race are:

- Encryption/KEM: CRYSTALS-KYBER [113], FRODOKEM [95], LAC [78], NEWHOPE [106], NTRU [123], NTRU PRIME [21], ROUND5 [56], SABER [44], THREE BEARS, [64].
- Signature schemes: CRYSTALS-DILITHIUM [82], FALCON [107], QTESLA [22].

In this thesis, we will focus mainly on NEWHOPE, QTESLA and to a lesser extend, KYBER. Those schemes are RLWE-based equivalents of the ElGamal encryption scheme and the Schnorr signature.

Chapter 3

Efficient KEMs Implementations on Embedded Devices

3.1 Preamble

This Chapter will discuss the work I did on high-speed implementations of lattice-based KEM on the Cortex-M4 microcontroller. It actually started with an idea I had for quite some time (early 2019 according to an email exchange with Gregor Seiler I found buried in my mailbox) which was to instanciate NEWHOPE with $q = 3329$. My feeling was that it is performance-wise lagging behind its Module-LWE sibling KYBER because the "partially splitting NTT" technique offering a larger choice in modulus was not used. Since my goal was to increase performances, I wanted to implement it on a platform which offered some point of comparison. The Cortex-M4 microcontroller was an obvious choice as it is the preferred platform for embedded devices in the NIST project and a full-fledged library to perform benchmarks and comparison called PQM4 is available [68, 70]. Since I had never worked on this platform before, I started with a smaller project which was to close a gap in implementation techniques between the early work of Alkim on NEWHOPE [13] and recent research in the context of the standardization process [31]. It basically consisted in revisiting the work of [13] with SIMD instructions. This led to a speed-up over their implementation and was integrated to the PQM4 library during summer 2019. Following my plan, I then implemented my idea of NEWHOPE with modulus 3329, with the expected success. While conversing with Matthias Kannwischer (who is managing PQM4), I learned that a team of researchers composed of Erdem Alkim, Yusuf Alper Bilgin and Murat Cenk was working on something similar. I discovered that they already had a paper accepted at Latincrypt 2019 [8] under the title "Compact and Simple RLWE Based Key Encapsulation Mechanism". Unfortunately, we were prior to the conference and the preprint was not available online. I contacted them shortly after to get a preliminary version and sadly realized that they already proposed a reduced modulus version of NEWHOPE (among other things) called NEWHOPE-COMPACT. On the bright side, their paper was coming with an AVX2 implementation but nothing on the embedded side. We then decided to team up and this led to a paper that went beyond my original idea by improving the Cortex-M4 implementations of KYBER and NEWHOPE (and hence the unpublished work that I did during summer) in addition to the first implementation of NEWHOPE-COMPACT on this platform [6].

3.2 Introduction

Cryptography is not created to live in a vacuum. At some point, it will be implemented in some service that uses it as a subroutine to protect data. For example, in a web browser, while the main task of the program is actually to display websites, sensitive communications are secured with some cryptographic algorithms before transiting on the network. Since users will understandably be unhappy if the programs they use are slow, the overhead of cryptography has to be considered. Hence, one of the most intuitive practical issue a cryptographic algorithm faces is pure efficiency in terms of speed. Of course, one can simply justify that security has a cost, but at some point, a trade-off between speed and security will be made and, if security levels (and bandwidth/storage) are similar, the fastest algorithm will be preferred. The good news is that cryptographic algorithms are often pretty lightweight relatively to other computing tasks. People nowadays are running all kind of graphical applications/games which take a lot of resources on their smartphone or computer. The additional cost of cryptography is thus often limited (but should not necessarily be neglected). Nevertheless, in the big world of small devices, the situation is different. Since the trend of the last couple of years seems to be to connect all everyday objects to the internet, the need for cryptographic implementations on embedded devices is very strong. Indeed, the set of connected objects like cars, refrigerators, watches, cameras, ..., that is now often called the Internet of Things, contains a lot of private data that must be secured. Furthermore, this type of systems are mostly very specialized, very limited in resources and designed to support only their base application. Thus, adding a cryptographic computation to the code might be greatly detrimental to performances. This is why it is required to study implementations tailored for commonly used architectures. This chapter will present contributions made to efficient implementations of lattice-based key encapsulation mechanisms on the Cortex-M4 platform. Beside the first part introducing needed background, it presents the result of an unpublished work that has been integrated in the PQM4 library [69] and a paper co-authored with Alkim et al. [6].

3.3 Polynomial Multiplication

The lattice-based KEMs we will study in this chapter are based on the ring learning with errors problem. The main computational tasks of such RLWE-based scheme are thus operations in the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle\phi_i\rangle$ of polynomials with coefficients in $\mathbb{Z}_q$ taken modulo the $i\text{-}th$ cyclotomic polynomial ϕ_i of degree $\varphi(i) = n$. Since the modulus q used in practice is quite small (usually less than 2^{32}), the addition is a straightforward cheap $\mathcal{O}(n)$ operation. The main bottleneck in terms of computation is the multiplication. Indeed, $\mathcal{R}_q$ is a quotient ring and thus has a specific multiplication law. Fortunately, since i is generally taken to be a power of two, the cyclotomic polynomial ϕ_i is of the form $X^n + 1$ and a simple procedure to multiply in $\mathcal{R}_q$ is to multiply the two elements in $\mathbb{Z}_q[X]$ and reduce modulo $X^n + 1$. This reduction step is actually very easy because one simply needs to replace all the X^n terms by -1 (since $X^n \equiv -1 \mod X^n + 1$). Thus, multiplying in the ring is not sensibly harder than a standard polynomial multiplication. Still, standard polynomial multiplication is more expensive than addition. The naive schoolbook algorithm runs in $\mathcal{O}(n^2)$ and for cryptographic parameters ($n = 512$ or $n = 1024$), the quadratic complexity hinders performances in practice. On the bright side, polynomial multiplication has already been extensively studied in the literature and algorithms such as Karatsuba/Toom-Cook are notably faster than the naive method. Still, for some rings,

even faster methods based on discrete Fourier transforms called NTTs are used.

3.3.1 NTT-based Multiplication

A Discrete Fourier Transform (DFT) over a finite field $\mathbb{Z}_q$, called a Number Theoretic Transform, is a function $\mathsf{NTT} : \mathbb{Z}_q^n \to \mathbb{Z}_q^n$ mapping a n-dimensional vector $\mathbf{x} = \{x_0, x_1, \ldots, x_{n-1}\}$ to an n-dimensional vector $\hat{\mathbf{x}} = \{\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1}\}$ such that

$$\hat{x}_k = \sum_{i=0}^{n-1} x_i \cdot \omega^{i \cdot k} \tag{3.1}$$

with ω a primitive n-th root of unity in $\mathbb{Z}_q$, which exists only if $q \equiv 1 \mod n$.

Its inverse, simply called the Inverse Number Theoretic Transform, is defined as $\mathsf{INTT} : \mathbb{Z}_q^n \to \mathbb{Z}_q^n, \hat{\mathbf{x}} = \{\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1}\} \mapsto \mathbf{x} = \{x_0, x_1, \ldots, x_{n-1}\}$ such that

$$x_k = \frac{1}{n} \sum_{i=0}^{n-1} \hat{x}_i \cdot \omega^{-i \cdot k} \tag{3.2}$$

It is well known that DFT can be used to compute the *cyclic convolution* of vectors ([42, 112]). Given two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^n$, the vector

$$\mathbf{c} = \mathsf{INTT}(\mathsf{NTT}(\mathbf{a}) \circ \mathsf{NTT}(\mathbf{b}))$$

with $\circ$ the component-wise multiplication of vectors, is the cyclic convolution of $\mathbf{a}$ and $\mathbf{b}$, that is to say the vector $\mathbf{c} \in \mathbb{Z}_q^n$ such that $c_i = \sum_{j+k \equiv i \bmod n} a_j \cdot b_k$. The reason why this property is very interesting is because if we interpret n-dimensional polynomials as vectors of their coefficients, the cyclic convolution between them corresponds to a multiplication in $\mathbb{Z}_q[X]/\langle X^n - 1 \rangle$. This is really close to the needed $\mathcal{R}_q$ multiplication. Actually, the $\mathcal{R}_q$ multiplication corresponds to what is called the *negative wrapped convolution* ([121] p.73) of $\mathbf{a}$ and $\mathbf{b}$ which introduces the -1 factor needed to perform the multiplication modulo $X^n + 1$. Fortunately, this can also be computed using the NTT. Let γ be a primitive $2n$-th root of unity such that $\gamma^2 = \omega$, $\mathbf{a}' = (a_0, \gamma \cdot a_1, \gamma^2 \cdot a_2, \ldots, \gamma^{n-1} \cdot a_{n-1})$ and $\mathbf{b}' = (b_0, \gamma \cdot b_1, \gamma^2 \cdot b_2, \ldots, \gamma^{n-1} \cdot b_{n-1})$, the negative wrapped convolution $\mathbf{c}$ of $\mathbf{a} = (a_0, a_1, a_2, \ldots, a_{n-1})$ and $\mathbf{b} = (b_0, b_1, b_2, \ldots, b_{n-1})$ is given by

$$\mathbf{c} = \left(1, \gamma^{-1}, \gamma^{-2}, \ldots, \gamma^{-(n-1)}\right) \circ \mathsf{INTT}(\mathsf{NTT}(\mathbf{a}') \circ \mathsf{NTT}(\mathbf{b}')). \tag{3.3}$$

The purpose of multiplying by powers of γ beforehand is to implicitly multiply by -1 coefficients that wrap around during the cyclic convolution, thanks to the equivalence $\gamma^n \equiv -1$. This yields a really efficient way to multiply polynomials in $\mathcal{R}_q$.

3.3.2 Fast Fourier Transform Algorithms

Even though we described how the NTT can be used to perform polynomial multiplication, it is not clear that this method is actually helpful. Indeed, looking at Equation 3.1, it seems that computing the whole NTT actually also requires $\mathcal{O}(n^2)$ operations in $\mathbb{Z}_q$. Luckily, various algorithms regrouped into the generic acronym FFT (Fast Fourier Transform) have been developed and reduced the complexity to $\mathcal{O}(n \log n)$ [36]. Since the right-hand side of 3.3 contains only (i)NTTs and cheap component-wise multiplications, we have a $\mathcal{O}(n \log n)$ algorithm for our polynomial multiplication in $\mathcal{R}_q$ as well.

Decimation In Time FFT

FFT algorithms use a divide-and-conquer strategy to efficiently computes DFTs. Let us start from Equation 3.1,

$$\hat{x}_k = \sum_{i=0}^{n-1} x_i \cdot \omega_n^{i \cdot k},$$

in which we write explicitly the order of the primitive root of unity, i.e. ω_n means that ω is a primitive n-th root of unity. If we separate the even and odd indices of the sum, we have

$$\begin{aligned}
\hat{x}_k &= \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \omega_n^{2i \cdot k} + \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \omega_n^{(2i+1) \cdot k} \\
&= \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \omega_n^{2i \cdot k} + \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \omega_n^{(2i) \cdot k} \\
&= \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \omega_{\frac{n}{2}}^{i \cdot k} + \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \omega_{\frac{n}{2}}^{i \cdot k}
\end{aligned} \tag{3.4}$$

since $\omega_n^2 = \omega_{\frac{n}{2}}$. Let us now set

$$A_k = \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \omega_{\frac{n}{2}}^{i \cdot k}$$

and

$$B_k = \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \omega_{\frac{n}{2}}^{i \cdot k},$$

we have

$$\hat{x}_k = A_k + \omega_n^k B_k \qquad \text{for } k = 0, 1, \ldots, \frac{n}{2} - 1$$

and

$$\begin{aligned}
\hat{x}_{k+\frac{n}{2}} &= \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \omega_{\frac{n}{2}}^{i \cdot (k+\frac{n}{2})} + \omega_n^{k+\frac{n}{2}} \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \omega_{\frac{n}{2}}^{i \cdot (k+\frac{n}{2})} \\
&= \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \omega_{\frac{n}{2}}^{i \cdot k} - \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \omega_{\frac{n}{2}}^{i \cdot k} \\
&= A_k - \omega_n^k B_k \qquad \text{for } k = 0, 1, \ldots, \frac{n}{2} - 1
\end{aligned} \tag{3.5}$$

which means that the Fourier transform of both x_k and $x_{k+\frac{n}{2}}$ can be easily computed from A_k and B_k. The interesting observation is that since k ranges from 0 to $\frac{n}{2} - 1$, A_k and B_k are themselves Fourier transforms of dimension $\frac{n}{2}$, which gives a nice recursive algorithm called the Decimation In Time (DIT) FFT. The two computations

- $\hat{x}_k \leftarrow A_k + \omega_n^k B_k$
- $\hat{x}_{k+\frac{n}{2}} \leftarrow A_k - \omega_n^k B_k$

are called *a butterfly*, due to the shape of the diagram in Figure 3.1. Since others FFT also use butterflies, this one is more specifically called a Cooley-Tukey butterfly due to the credited authors of this method [37].

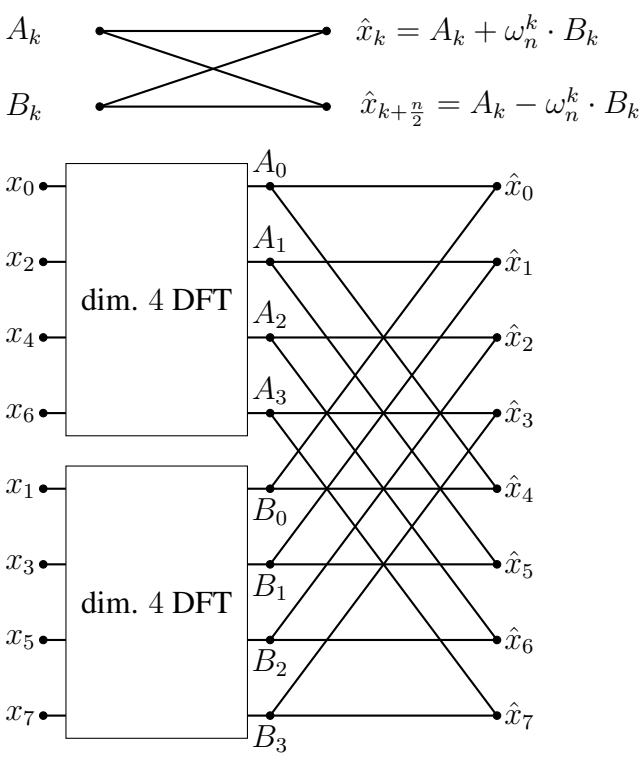

Figure 3.1: The Cooley-Tukey butterfly and its usage to perform a full DIT FFT recursively.

Decimation In Frequency FFT

An alternative technique, which result in a slightly different algorithm is to separate the odd/even indices of the $\hat{\mathbf{x}}$ vector instead of the $\mathbf{x}$ vector. This method is called decimation in frequency (DIF) [1]. Recall Equation 3.1,

$$\hat{x}_k = \sum_{i=0}^{n-1} x_i \cdot \omega_n^{i \cdot k},$$

with primitive n-th root of unity ω_n. Splitting the sum in two parts,

$$\begin{aligned}
\hat{x}_k &= \sum_{i=0}^{n-1} x_i \cdot \omega_n^{i \cdot k} \\
&= \sum_{i=0}^{\frac{n}{2}-1} x_i \cdot \omega_n^{i \cdot k} + \sum_{i=\frac{n}{2}}^{n-1} x_i \cdot \omega_n^{i \cdot k} \\
&= \sum_{i=0}^{\frac{n}{2}-1} x_i \cdot \omega_n^{i \cdot k} + \sum_{i=0}^{\frac{n}{2}-1} x_{i+\frac{n}{2}} \cdot \omega_n^{(i+\frac{n}{2}) \cdot k} \\
&= \sum_{i=0}^{\frac{n}{2}-1} \left(x_i + x_{i+\frac{n}{2}} \cdot \omega_n^{k \cdot \frac{n}{2}} \right) \cdot \omega_n^{k \cdot i} \qquad \text{for } k = 0, 1, \ldots, n-1,
\end{aligned} \tag{3.6}$$

[1] This nomenclature should be clear to the reader familiar with Fourier transforms but is somewhat irrelevant in our context.

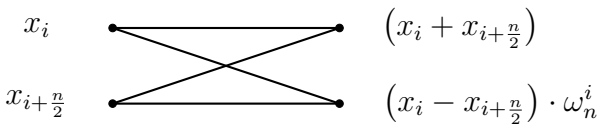

Figure 3.2: Gentleman-Sande Butterfly

we separate the even values of k from the odd ones. For the even values $k = 2j$,

$$\begin{aligned}\hat{x}_{2j} &= \sum_{i=0}^{\frac{n}{2}-1} \left(x_i + x_{i+\frac{n}{2}} \cdot \omega_n^{j \cdot n}\right) \cdot \omega_n^{2j \cdot i} \\ &= \sum_{i=0}^{\frac{n}{2}-1} \left(x_i + x_{i+\frac{n}{2}}\right) \cdot \omega_{\frac{n}{2}}^{j \cdot i} \qquad \text{for } k = 0, 1, \ldots, \frac{n}{2} - 1. \end{aligned} \tag{3.7}$$

For the odd values $k = 2j + 1$,

$$\begin{aligned}\hat{x}_{2j+1} &= \sum_{i=0}^{\frac{n}{2}-1} \left(x_i + x_{i+\frac{n}{2}} \cdot \omega_n^{(2j+1) \cdot \frac{n}{2}}\right) \cdot \omega_n^{(2j+1) \cdot i} \\ &= \sum_{i=0}^{\frac{n}{2}-1} \left(x_i + x_{i+\frac{n}{2}} \cdot \omega_n^{\frac{n}{2}}\right) \cdot \omega_n^{i} \cdot \omega_{\frac{n}{2}}^{j \cdot i} \\ &= \sum_{i=0}^{\frac{n}{2}-1} \left(\left(x_i - x_{i+\frac{n}{2}}\right) \cdot \omega_n^{i}\right) \cdot \omega_{\frac{n}{2}}^{j \cdot i} \qquad \text{for } k = 0, 1, \ldots, \frac{n}{2} - 1. \end{aligned} \tag{3.8}$$

Again, from 3.7 and 3.8, we expressed the size n transform as a combination of two size $\frac{n}{2}$ transforms, which leads to a recursive FFT algorithm. The dividing part before the recursion is to compute the values $\left(x_i + x_{i+\frac{n}{2}}\right)$ and $\left(x_i - x_{i+\frac{n}{2}}\right) \cdot \omega_n^i$, this operation is called a *Gentleman-Sande butterfly* (Figure 3.2) [57]. The combination step after solving the size $\frac{n}{2}$ subproblems is trivial as they respectively give the even and odd values of the transform.

NTT used in lattice-based cryptography

Really efficient iterative algorithms were derived from the recursive procedures described by the DIT and DIF methods, see [36] for an extensive survey. In the lattice-based cryptography literature, several approaches with similar performances have flourished over the years. For example, NEWHOPE straightforwardly implements the multiplication in $\mathcal{R}_q$ from Equation 3.3 using a DIF FFT while some other works merged the multiplication with $2n$-th roots of unity directly into the transform. One common issue with in-place computations of the NTT is that the output vector is in bitreversed order, which means that a coefficient at index i in the output vector actually corresponds to the coefficient at index $\mathsf{BitRev}(i)$ in the NTT of the input vector, with $\mathsf{BitRev}(.)$ a function reversing the binary representation of its input (see Table 3.1). In NEWHOPE, the bitreversal step is explicitly performed on polynomials. In KYBER, the NTT and INTT algorithms used are designed such that no bitreversal is needed. Indeed, its NTT maps polynomials in normal order to polynomials in bitreversed order while its INTT maps polynomials in bitreversed order to polynomials in normal order. Hence, when computing a multiplication

$c = \mathsf{INTT}(\mathsf{NTT}(a) \circ \mathsf{NTT}(b))$, there is no need for a bitreversal since the INTT "undoes" the scrambling of the NTT.

i	$\mathsf{Bin}(i)$	$\mathsf{RevBin}(i)$	$\mathsf{BitRev}(i)$
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Table 3.1: Bitreversal table on 3 bits

The algebraic approach

An algebraic approach to the computation of the NTT is presented in [114]. Multiplying polynomials by computing an expression of the form $\mathsf{IDFT}(\mathsf{DFT}(a) \circ \mathsf{DFT}(b))$ is actually two evaluations followed by an interpolation. Indeed, it is clear from Equation 3.1 that the DFT is evaluating n times the polynomial with coefficients x_i at different roots of unity. The pointwise multiplication $\circ$ is then a multiplication of the evaluated points between them and finally, the inverse DFT interpolates the n points to a polynomial. To multiply two elements in the ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$, the trick is to evaluate the polynomial at the roots of $X^n + 1$ in $\mathbb{Z}_q$ such that the multiples of this polynomial disappear during the evaluation. Since, in RLWE, $X^n + 1$ is the $2n$-th cyclotomic polynomial, n is a power of two and its roots are the $n = \varphi(2n)$ values $\gamma, \gamma^3, \ldots, \gamma^{2n-1}$ with γ a primitive $2n$-th root of unity in $\mathbb{Z}_q$. Hence, the NTT can be seen as the computation of the isomorphism

$$\mathbb{Z}_q[X]/\langle X^n + 1\rangle \to \prod_{i\in\{1,3,\ldots,2n-1\}} \mathbb{Z}_q[X]/\langle X - \gamma^i\rangle : p \mapsto \left(p(\gamma), p(\gamma^3), \ldots, p(\gamma^{2n-1})\right) \tag{3.9}$$

It can be efficiently computed using the Chinese Remained Theorem. The idea is to map the ring

$$\mathbb{Z}_q[X]/\langle X^n - \gamma^n\rangle$$

to

$$\mathbb{Z}_q[X]/\langle X^{n/2} - \gamma^{n/2}\rangle \times \mathbb{Z}_q[X]/\langle X^{n/2} + \gamma^{n/2}\rangle$$

by computing the straightforward CRT map

$$p \mapsto (p \bmod X^{n/2} - \gamma^{n/2}, p \bmod X^{n/2} + \gamma^{n/2}).$$

Since $X^{n/2} + \gamma^{n/2} = X^{n/2} - \gamma^{n+n/2}$ and n is a power of two, the same map can be computed again on both components until reaching a product of rings in which the base multiplication is cheap.

The core operation, which is to reduce a polynomial p modulo $X^{n/2} \pm \zeta$ (for ζ a specific power of γ) corresponds in fact in computation of Cooley-Tukey butterflies. The computation of the NTT consists in applying $n/2$ butterflies to pairs of coefficients of the whole polynomial iteratively between 1 and $\log_2 n$ times, each iteration being referred

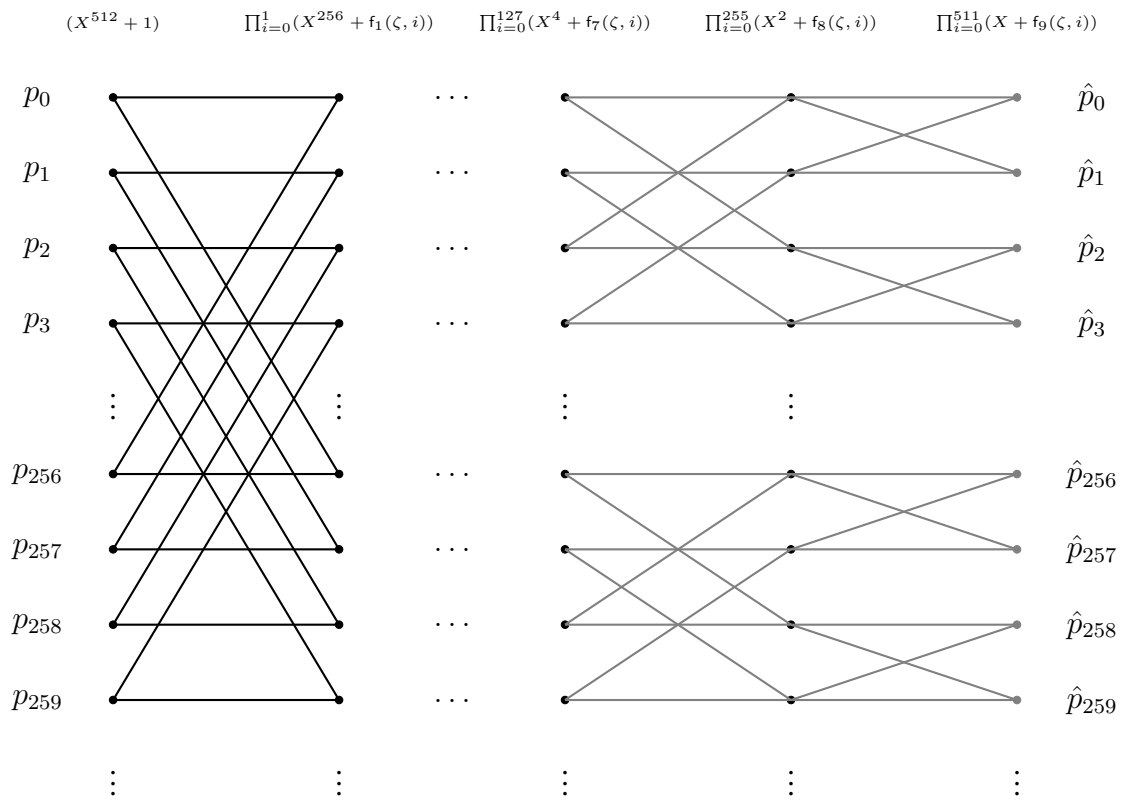

Figure 3.3: Full NTT on a dimension 512 polynomial. The function $f_j(\zeta, i) = \zeta^{\mathsf{BitRev}(2^j-1+i)}$ selects the correct root to compute the isomorphism. All those roots are usually precomputed and correctly ordered in a table. Techniques to reduce q skip some levels: for example, using $q = 3329$ requires to skip the two lasts (gray) layers.

to as a layer. Figure 3.3 pictures a full NTT consisting of 9 layers mapping the ring $\mathbb{Z}_q[X]/\langle X^{512}+1\rangle$ to a product of rings of the form $\mathbb{Z}_q[X]/\langle X-\zeta\rangle$. In KYBER, the NTT is actually stopped earlier because $\mathbb{Z}_q$ does not offer high order enough roots of unity to compute all of the layers. This means that the NTT itself is less expensive but the base operation (which is pointwise multiplication for a full NTT) in the product of rings is more computationally intensive. This base operation is polynomial multiplication in rings of the form $\mathbb{Z}_q[X]/\langle X^a - b\rangle$.

3.4 Lattice-Based Key Exchange/Key Encapsulation

We focus now on NEWHOPE and KYBER, two candidates in the NIST standardization project. They both stem from the same line of research studying RLWE-based equivalents of Diffie-Hellman and ElGamal. Basically, the idea is to replace values $y = g^x$ (or $y = g \circ x$ for an abstract group) by RLWE samples of the form $y = a \cdot x + e$. For example, in the Diffie-Hellman key exchange, Alice picks a, Bob picks b, they respectively send g^a and g^b and both use their secret to compute the shared key g^{ab}. In the RLWE version, Alice picks s, Bob picks s', they respectively send $a \cdot s + e$ and $a \cdot s' + e'$ and both use their secret to compute

- $k = s \cdot (a \cdot s' + e') = a \cdot s' \cdot s + e' \cdot s$ for Alice and,
- $k' = s' \cdot (a \cdot s + e) = a \cdot s \cdot s' + e \cdot s'$ for Bob.

They both end up with different values (which is quite problematic for a key exchange!) but the difference between k and k' (which is $e' \cdot s - e \cdot s'$) is really small because s, s', e and e' are all sampled from a narrow error distribution, for example a centered binomial of low variance (with respect to q). Hence, one can hope to define a reconciliation or decoding procedure that maps a set to a smaller set such that values that are close in the input set are mapped to the same output. This type of procedure is basically a rounding, for example, one can drop the least significant bits of each value. Recall that in the case of RLWE, we are working with polynomials, thus the rounding should be applied to each of the coefficients, which are elements of $\mathbb{Z}_q$. Unfortunately, rounding procedures do not work for corner cases. For example, if we define the simple rounding function (we assume q is even for the sake of simplicity)

$$f(x) = \begin{cases} 0 \text{ if } 0 \leq x < q/2 \\ 1 \text{ if } q/2 \leq x < q \end{cases}$$

and trivially extend it to a function rounding polynomials by applying f independently to each coefficient, if a coefficient of Alice a_i is slightly below $q/2$ and the corresponding coefficient of Bob $b_i = a_i + \delta_i$ slightly over, the reconciliation fails. Since this scenario is very likely with large polynomials, RLWE key exchange add another message to the protocol called the reconciliation vector that helps the other party correctly rounding without revealing sensitive information on the shared key to the adversary, see [100] and Section 5.6. Similarly to the discrete logarithm case, the DH-like RLWE key exchange can be converted to an ElGamal-like public-key encryption scheme. This is basically what has been done with NEWHOPE and KYBER.

3.4.1 NewHope

NEWHOPE is the generic name for two KEMs algorithms called NEWHOPE-CPA-KEM and NEWHOPE-CCA-KEM. They are both constructed from the lattice-based public-key encryption scheme called NEWHOPE-CPA-PKE, which is the main component. Since NEWHOPE-CPA-KEM and NEWHOPE-CCA-KEM both use standard generic techniques to convert a semantically secure public-key encryption scheme into passively and actively secure key encapsulation mechanisms, all the lattice-based parts and, thus, all the optimizations proposed reside in NEWHOPE-CPA-PKE.

NEWHOPE-CPA-PKE

NEWHOPE-CPA-PKE is a PKE based on a scheme previously known in the literature under the name NEWHOPE-SIMPLE [11] which is an encryption-based variant of the RLWE-based key exchange NEWHOPE-USENIX [12, 28]. It is parameterized by the integers n, q and k defining the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and the binomial distribution $\psi_k = \sum_{i=1}^{k} b_i - b'_i$ with each b_i and b'_i sampled uniformly at random from $\{0, 1\}$. It is an ElGamal-like variant of the lattice-based key exchange loosely described in Section 3.4. The algorithm is explicitly give in Figure 3.4, it contains a certain amount of subroutines that we simply functionally outline:

- GenA: Samples a uniformly random element in $\mathcal{R}_q$. Uses SHAKE256 internally to generate random values.

Algorithm 1 NH-CPA-PKE.Gen

Output: public key $pk = (\hat{b}', \rho)$
Output: secret key $sk = \hat{s}$

1: $seed \xleftarrow{r} \{0, \cdots, 255\}^{32}$
2: $\rho, \sigma \leftarrow \mathsf{SHAKE256}(64, seed)$
3: $\hat{a} \leftarrow \mathsf{GenA}(\rho)$
4: $s \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(\sigma, 0))$
5: $e \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(\sigma, 1))$
6: $\hat{b} \leftarrow \hat{a} \circ \mathsf{NTT}(s) + \mathsf{NTT}(e)$
7: **return** $pk = (\hat{b}, \rho), sk = \hat{s}$

Algorithm 2 NH-CPA-PKE.Encrypt

Input: public key $pk = (\hat{b}, \rho)$
Input: message $\mu \in \{0, 1\}^{256}$
Input: seed $coin \in \{0, \cdots, 255\}^{32}$
Output: ciphertext $(\hat{u}', h)$

1: $\hat{a} \leftarrow \mathsf{GenA}(\rho)$
2: $s' \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(coin, 0))$
3: $e' \leftarrow \mathsf{PolyBitRev}(\mathsf{Sample}(coin, 1))$
4: $e'' \leftarrow \mathsf{Sample}(coin, 2)$
5: $\hat{t} \leftarrow \mathsf{NTT}(s')$
6: $\hat{u} \leftarrow \hat{a} \circ \hat{t} + \mathsf{NTT}(e')$
7: $v' \leftarrow \mathsf{INTT}(\hat{b} \circ \hat{t}) + e'' + \mathsf{Encode}(\mu)$
8: **return** $c = (\hat{u}, \mathsf{Compress}(v'))$

Algorithm 3 NH-CPA-PKE.Decrypt

Input: ciphertext $c = (\hat{u}, h)$
Input: secret key $sk = \hat{s}$
Output: message $\mu \in \{0, \cdots, 255\}^{32}$

1: $v' \leftarrow \mathsf{Decompress}(h)$
2: **return** $\mu = \mathsf{Decode}(v' - \mathsf{INTT}(\hat{u} \circ \hat{s}))$

Figure 3.4: NEWHOPE-CPA-PKE, the RLWE-based ElGamal-like main component of NEWHOPE-CPA-KEM and NEWHOPE-CCA-KEM

- PolyBitRev: Reorders the coefficient of the polynomial with a bitreversal using a precomputed table.
- Sample: Samples from the binomial distribution ψ_k. Uses SHAKE256 internally to generate random values with the seed σ and an integer acting as a domain separator.
- Encode: Encode each bit of the message in $\lfloor n/256 \rfloor$ coefficients of a polynomial in $\mathcal{R}_q$. For a given bit b of the message, each of those coefficients is set to $b \cdot \frac{q}{2}$. The output is a polynomial of dimension n with coefficients in $\{0, \frac{q}{2}\}$ that contains $\lfloor n/256 \rfloor$ copies of the message.
- Decode: Recovers the message from a noisy encoded polynomial. The idea is, for each bit encoded as a value between 0 and $q-1$, to sum the distance to $q/2$ of the $\lfloor n/256 \rfloor$ corresponding coefficients and output 1 if the sum is greater than $\lfloor n/256 \rfloor \cdot q/4$. For example, with $n = 256$ (which would not be secure !), it simply sets the bit to 0 if the distance with $q/2$ is over $q/4$, which means that the encoded bit is in $[0, q/4[\cup [3q/4, q-1]$.
- Compress and Decompress: Modulus switching from q to 8 to compress parts of the ciphertext, reducing bandwidth usage. The decompression function switches back to values between 0 and $q-1$. This is analogous to discarding low bits of the values. Of course, some information is lost in the process but the decoding procedure can tolerate quite some amount of noise.

A full formal description of those functions can be found in the technical specification document submitted to the NIST project [119]. Those functions are not an optimization target because either they are pretty lightweight, or their performance depends heavily on the hash function which is independently optimized. The speedup presented in this chapter will actually be improvements inside the NTT.

Algorithm 4 NH-CPA-KEM.Gen

1: $pk, sk \leftarrow$ NH-CPA-PKE.Gen()
2: **return** pk, sk

Algorithm 5 NH-CPA-KEM.Decaps

Input: ciphertext c
Input: secret key sk

1: $k' \leftarrow$ NH-CPA-PKE.Decrypt(c, sk)
2: **return** $k =$ SHAKE256$(32, k')$

Algorithm 6 NH-CPA-KEM.Encaps

Input: public key pk

1: $coin \xleftarrow{r} \{0, \ldots, 255\}^{32}$
2: $k'||coin' \leftarrow$ SHAKE256$(64, coin)$
3: $c \leftarrow$ NH-CPA-PKE.Encrypt$(pk, k'; coin')$
4: $k \leftarrow$ SHAKE256$(32, k')$
5: **return** (c, k)

Figure 3.5: NEWHOPE-CPA-KEM

NewHope CPA-KEM and CCA-KEM

The conversion from an IND-CPA public-key encryption scheme is basically the one presented in Definition 2.2.5 and is formally shown in Figure 3.5. However, it is more complicated to switch from a CPA-PKE to a CCA-KEM. The usual technique is to apply a black-box transformation called the Fujisaki-Okamoto transform [54]. Sadly, it is proven secure in the classical random oracle model but does not work for a quantum adversary. In [66], Hofheinz, Hövelmanns, and Kiltz described a couple of alternative transformations that are suitable for schemes with non-zero failure probabilities in the quantum case. One of them is used to obtain NEWHOPE-CCA-KEM from NEWHOPE-CPA-PKE and can be found in Figure 3.6.

Parameter sets

The parameters of NEWHOPE are:

- The dimension n,
- The modulus q,
- The parameter of the error distribution k.

The most sensitive of them in terms of security is the dimension n. One looking for a significant increase in hardness of the underlying RLWE instance will switch to a higher dimension. Since NEWHOPE is defined with a cyclotomic polynomial of the form X^n+1, n should be a power of two. The security also depends, to a lower extend, on the ratio between q and k. Indeed, intuitively, if k is very small in comparison to q, the underlying RLWE instance is easier to solve since the errors are small. Finding a good trade-off between those three values is complex and requires to do a careful analysis of the current best cryptanalytic techniques. The method used by the authors of NEWHOPE is described in the specification document and will not be explained here. Another practical constraint for choosing parameters is that the NTT requires a prime q and the existence of $2n$-th roots of unity in $\mathbb{Z}_q$. For them to exist, the relation $q \equiv 1 \mod 2n$ should hold. Actually the modulus of NEWHOPE is chosen as the smallest prime for which this holds for $n \in \{512, 1024\}$, which are the first possible dimensions of the ring that offer reasonable security estimates. The proposed parameters can be found in Table 3.2. NIST security strength categories correspond to security levels defined by NIST in the framework of the

Algorithm 7 NH-CCA-KEM.Gen

1: $pk, sk \leftarrow$ NH-CPA-PKE.Gen()
2: $s \leftarrow \{0, \ldots, 255\}^{32}$
3: **return** $pk, \hat{sk} = sk||pk||\mathsf{SHAKE256}(32, pk)||s)$

Algorithm 8 NH-CCA-KEM.Encaps

Input: public key pk

1: $coin \xleftarrow{r} \{0, \ldots, 255\}^{32}$
2: $\mu \leftarrow \mathsf{SHAKE256}(32, coin)$
3: $k'||coin||d \leftarrow \mathsf{SHAKE256}(96, \mu||\mathsf{SHAKE256}(32, pk))$
4: $c \leftarrow$ NH-CPA-PKE.Encrypt$(pk, \mu, coin')$
5: $k \leftarrow \mathsf{SHAKE256}(32, k'||\mathsf{SHAKE256}(32, c||d))$
6: **return** $(\hat{c} = c||d, k)$

Algorithm 9 NH-CCA-KEM.Decaps

Input: ciphertext $\hat{c}$
Input: secret key $\hat{sk}$

1: $c||d \leftarrow \hat{c}$
2: $sk||pk||h||s \leftarrow \hat{sk}$
3: $\mu' \leftarrow$ NH-CPA-PKE.Decrypt(c, sk)
4: $k'||coin''||d' \leftarrow \mathsf{SHAKE256}(96, \mu'||h)$
5: $b_1 \leftarrow$ NH-CPA-PKE.Encrypt$(pk, \mu'; coin'') == c$
6: $b_2 \leftarrow d == d'$
7: $fail \leftarrow \neg(b_1 \wedge b_2)$
8: $k'_0 = k'$
9: $k'_1 = s$
10: **return** $k = \mathsf{SHAKE256}(32, k'_{fail}||\mathsf{SHAKE256}(32, c||d))$

Figure 3.6: NEWHOPE-CCA-KEM

Parameter Set	NEWHOPE512	NEWHOPE1024
Dimension n	512	1024
Modulus q	12289	12289
Noise Parameter k	8	8
Decryption error probability	2^{-213}	2^{-216}
Claimed post-quantum bit security	101	233
NIST Security Strength Category	1	5

Table 3.2: Parameter sets of NEWHOPE

standardization process, 1 is the lowest and 5 is the highest. The security levels 1,3 and 5 can be tough to correspond to a symmetric bit-security of 128, 192 and 256 bits.

3.4.2 Kyber

We move on to the next NIST candidate improved by our work: KYBER [29, 118]. Coming from the same family as NEWHOPE and FRODO, it is a promising representative for LWE-based encryption/KEM. Since it shares many similarities with NEWHOPE that has just been depicted in the previous section, we will just give a light description of the cryptosystem, major differences with NEWHOPE and information relevant to our contribution.

Module Learning With Errors

The first obvious difference between NEWHOPE and KYBER is that KYBER is based on a variant of Learning With Errors called Module Learning With Errors (MLWE) [75]. Depending on the parameters used, this problem is exactly LWE, exactly RLWE, or something in-between. It is parametrized by a modulus q, a dimension n, a ring $\mathcal{R}_q$, which, similarly to RLWE, will usually be $\mathbb{Z}_q[X]/\langle X^n+1\rangle$, an integer k and an error distribution χ over $\mathcal{R}_q$.

Definition 36. *(search-MLWE) For a fixed* $\mathbf{s} \in \mathcal{R}_q^k$*, given* m *samples* $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i)$ *with the* $\mathbf{a}_i$ *sampled uniformly from* $\mathcal{R}_q^k$ *and the* e_i *sampled from* χ*, find* $\mathbf{s}$*.*

Definition 37. *(decision-MLWE) Given* m *samples* $(\mathbf{a}_i, b_i) \in \mathcal{R}_q^k \times \mathcal{R}_q$ *with each* $\mathbf{a}_i$ *sampled uniformly at random, decide if the* b_i *were also sampled uniformly at random or if they were computed as* $b_i = \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i$ *for a fixed* $\mathbf{s}$ *and each* e_i *sampled from* χ*.*

Basically, while LWE asked to undo a noisy matrix-vector multiplication over entries in $\mathbb{Z}_q$ and RLWE a noisy polynomial multiplication in $\mathcal{R}_q$, MLWE asks to undo a noisy matrix-vector multiplication over entries in $\mathcal{R}_q$. If k is set to 1, the dot product in $\mathcal{R}_q$ collapses to a polynomial multiplication in $\mathcal{R}_q$ and this is simply a RLWE instance. If n is set to 1, $\mathcal{R}_q$ collapses to $\mathbb{Z}_q$ and this is a LWE instance. Thus, MLWE is a modular problem which links the two problems (LWE and RLWE) defined in Section 2.3.3 and cryptosystems can be defined in a modular fashion to base their security on each problem depending on the parameters used.

Algorithm 10 KYBER-CPA-PKE.Gen

Output: public key $pk = (\hat{\mathbf{b}}, \rho)$
Output: secret key $sk = \hat{\mathbf{s}}$

1: $seed \xleftarrow{r} \{0, \cdots, 255\}^{32}$
2: $\rho, \sigma \leftarrow \mathsf{SHAKE256}(64, seed)$
3: $\hat{\mathbf{A}} \leftarrow \mathsf{GenMatrixA}(\rho)$
4: $\mathbf{s} \leftarrow \mathsf{SampleVec}(\sigma, 0)$
5: $\mathbf{e} \leftarrow \mathsf{SampleVec}(\sigma, 1)$
6: $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{s}) + \mathsf{NTT}(\mathbf{e})$
7: **return** $pk = (\hat{\mathbf{b}}, \rho), sk = \hat{\mathbf{s}}$

Algorithm 11 KYBER-CPA-PKE.Decrypt

Input: ciphertext $c = (\mathbf{u}', h)$
Input: secret key $sk = \hat{\mathbf{s}}$
Output: message μ as a polynomial in $\mathcal{R}_q$

1: $\mathbf{u} \leftarrow \mathsf{Decompress}(\mathbf{u}')$
2: $v' \leftarrow \mathsf{Decompress}(h)$
3: **return** $\mu = v' - \mathsf{INTT}(\hat{\mathbf{s}}^T \circ \mathsf{NTT}(\mathbf{u}))$

Algorithm 12 KYBER-CPA-PKE.Encrypt

Input: public key $pk = (\hat{\mathbf{b}}, \rho)$
Input: message μ as a polynomial in $\mathcal{R}_q$
Input: seed $coin \in \{0, \cdots, 255\}^{32}$
Output: ciphertext $(\hat{\mathbf{u}}', h)$

1: $\hat{\mathbf{A}} \leftarrow \mathsf{GenMatrixA}(\rho)$
2: $\mathbf{s}' \leftarrow \mathsf{SampleVec}(coin, 0)$
3: $\mathbf{e}' \leftarrow \mathsf{SampleVec}(coin, 1)$
4: $e'' \leftarrow \mathsf{SampleVec}(coin, 2)$
5: $\hat{\mathbf{t}} \leftarrow \mathsf{NTT}(\mathbf{s}')$
6: $\mathbf{u} \leftarrow \mathsf{INTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{t}}) + \mathbf{e}'$
7: $v' \leftarrow \mathsf{INTT}((\hat{\mathbf{b}}^T \circ \hat{\mathbf{t}}) + e'' + \mu$
8: **return** $(\mathsf{Compress}(\mathbf{u}), \mathsf{Compress}(v'))$

Figure 3.7: KYBER-CPA-PKE

Kyber-CPA-PKE

Like NEWHOPE, KYBER is an IND-CCA Key Encapsulation Mechanism KYBER-CCA-KEM constructed from a CPA Public-Key Encryption scheme KYBER-CPA-PKE using a variant of the Fujisaki-Okamoto transform. Figure 3.7 gives a high-level overview of the underlying PKE. It is simple to show its semantic security under the decision-MLWE assumption. It uses the following subroutines:

- GenMatrixA: Generates a $k \times k$ matrix with entries in $\mathcal{R}_q$ using SHAKE256 on a seed given as input.
- SampleVec: Samples a k dimensional vector with entries from a binomial distribution using SHAKE256 on a seed and a nonce given as input.
- Compress and Decompress: Same purpose as the NEWHOPE equivalent, are defined in a way such that they are also used to scale the message during encryption and decode the noisy message during decryption. [2]

Parameter sets

KYBER fixes the dimension of the ring $\mathcal{R}_q$ to 256 across all parameter sets. This choice permits to naturally encode a 256-bit message, which is the key size mainly used for a post-quantum PKE used as KEM, in a single ring element, using one coefficient per bit. The mains parameters are thus :

- k: The size of the vectors and of the square matrix $\hat{\mathbf{A}}$.

[2] Figure 3.7 already assumes that the message is rightly encoded to avoid overloading the algorithms.

Parameter Set	KYBER512	KYBER768	KYBER1024
Rank k	2	3	4
Dimension $k \cdot n$	512	768	1024
Modulus q	3329	3329	3329
Noise Parameter η	2	2	2
Compression Parameter d_u, d_v	10,3	10,4	11,5
Decryption error probability	2^{-178}	2^{-164}	2^{-174}
Claimed post-quantum bit security	100	164	230
NIST Security Strength Category	1	3	5

Table 3.3: Parameter sets of KYBER

- η: The parameter of the binomial distribution $B_\eta = \sum_{i=1}^{\eta} b_i - b'_i$, with b_i and b'_i in $\{0, 1\}$, used to sample the error and secret vectors [3].
- d_u, d_v: The amount of compression applied to the ciphertext. Setting the compression to high would lead to non-negligible probability of failure during decryption.

All of them, together with estimated correctness and security estimations are resumed in Table 3.3. Some interesting remarks about those parameters:

- Since the dimension of the underlying LWE problem is actually given by $k \cdot n$, KYBER can actually be instanciated with a dimension $768 = 3 \cdot 256$ problem. To keep power of two cyclotomics as defining polynomials, NEWHOPE is stuck with 512 and 1024 without any possible intermediate value.
- The values q and η (and implicitly n) are the same in each set, which means that k really defines the security/performances trade-off.
- The modulus q is a prime but is not congruent to 1 modulo $2n$, which means that a full NTT on the ring elements cannot be performed.
- Since the modulus has been reduced, the support of the error distribution was also reduced (this mean lower bandwidth and less random number generation needed) while still offering decent security. This virtuous circle could be somewhat continued but a too narrow distribution might lead to more efficient cryptanalysis [4].

3.5 Fast NTT for Fast Implementations of NewHope and Kyber on Cortex-M4

As indicated earlier, polynomial multiplication is the heaviest operation of lattice KEMs such as KYBER or NEWHOPE. Actually, in an optimized implementation, most of the time is eventually spent in the hash functions used to generate random numbers, but the speed issues of such functions is not specific to lattice-based cryptography and the choice of the hash function is arbitrary. Candidates for the NIST project chose SHAKE but other

[3] This is actually the same definition as the distribution for NEWHOPE but decided to keep different notations to be consistent with the specifications of both schemes.

means of random numbers generation can be considered. On the other hand, polynomial multiplication will always be needed. Since NTTs give an elegant way to perform those multiplications when the parameters are chosen such that those transforms exist, all the optimization work boils down to speed them up.

3.5.1 Efficient Reduction Algorithms

Beside actually efficiently computing the NTT using FFT algorithms described in Section 3.3.2, another difficulty lies in the fact that computations must be performed in the field $\mathbb{Z}_q$. While this might seems totally trivial at first sight since every programmer knows the modulus operator, reducing modulo q in an *efficient* and *constant time* manner is a bit more challenging. Furthermore, always reducing modulo the same q offers some possible speed-ups over using the % operator.

Lazy reductions

Mathematically, $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ is composed of the q cosets $\{x + q\mathbb{Z}\}_{0 \leqslant x \leqslant q-1}$ and rings operations are defined on those sets. In practice, one chooses an element of each coset to represent it. The two obvious choices are to take the integers between 0 and $q - 1$ (the x in the definition above), often called the *canonical representatives*, or between $\lfloor q/2 \rfloor$ and $\lfloor (q-1)/2 \rfloor$, often called the *centered representatives*. Operations are then performed in the integers and results are mapped to a representative by subtracting an appropriate multiple of q, this map is called a (full) reduction. The % operator computes reductions to representatives that are language or operand dependent. However, the choice of the representative does not matter at all for the computations to be correct in $\mathbb{Z}_q$. Thus, a technique used to minimize the amount of computation is to simply avoid reductions for intermediate results. For example, instead of computing `((a+b)%q + c)%q`, one computes `(a+b+c)%q`. Of course, the bulk of the work is not to avoid reducing but to estimate how many operations can be safely performed without having a result larger than what the datatype we use can hold. This technique is called *lazy reductions*.

Barrett reduction

A well-known technique to reduce an integer modulo q is to compute

$$r \leftarrow x - \lfloor \frac{x}{q} \rfloor \cdot q.$$

It is not really lightweight since it requires a division by q, however, multiplying by an approximation of the constant $\frac{1}{q}$ will still give a result in the right coset since the values can only differ from some multiple of q. The better the approximation is, the closer the result will be from a full reduction. Unfortunately, as q is a positive integer, $\frac{1}{q}$ is smaller than 1 would require floating-point arithmetic. Working only with integers, the hope comes from the fact that there is one type of integer division which is easy on any reasonable hardware: division by a power of two. Thus, the idea is to pick two integers s, c such that $\frac{1}{q} \approx c/2^s$ to approximate the constant. The reduction can then be simply computed as `r = x - ((x*c)>>s)*q`. Setting c to the integer closest to $2^s/q$, the only real parameter is s. An higher s gives a better approximation but one should be careful to avoid overflow on the product of x by c. The Barrett reduction was described by Barrett in [17].

Montgomery reduction

The last reduction technique we present is due to Peter Montgomery [93], who sadly passed away a few weeks before writing this section. It uses the fact that if, for an integer M (coprime with q) called the Montgomery constant, reductions and divisions by M are easy, there exists a really efficient procedure to compute a $0 \leqslant r \leqslant q-1$ such that $r \equiv x \cdot M^{-1} \mod q$, given x. This procedure is called a Montgomery reduction ($\mathsf{MRed}()$) and is given in Algorithm 13. Of course, picking M a power of two is the obvious choice here since divisions and reductions modulo M will thus be very fast using shifts and masks. Nevertheless, it is not directly clear that such a reduction is helpful. Indeed, to retrieve the reduction of x modulo q from r, one has to multiply r by M and ... reduce modulo q, which was the original problem! Actually, the Montgomery reduction algorithm shines when several arithmetic operations have to be performed modulo q. Let us consider two values a and b reduced modulo q. The values $a \cdot M \mod q$ and $b \cdot M \mod q$ are said to be the *Montgomery domain representatives* (or Montgomery forms) of a and b. Arithmetic in Montgomery domain can be efficiently performed. Indeed, $aM \pm bM \equiv (a \pm b)M \mod q$ and $\mathsf{MRed}(aM \cdot bM) = \mathsf{MRed}(ab \cdot M^2) = (ab \cdot M) \mod q$. Hence, when a lot of arithmetic operations are needed, converting all values in Montgomery form, computing everything in Montgomery domain and converting back to the normal domain at the end is more efficient than using usual reductions in all computations. Another really useful application is multiplication of a value v by a precomputed constant k. Indeed, instead of storing k, one stores $k' = kM \mod q$ and simply computes $\mathsf{MRed}(v \cdot k')$ to reduce $v \cdot k$ modulo q. This is really helpful in the NTT since precomputed roots of unity can be stored in Montgomery domain.

Algorithm 13 Montgomery reduction

Input: $x \in \{0, \cdots, Mq-1\}$
Input: Precomputed $q_{inv} \in \{0, \ldots, q-1\}$ such that $q_{inv} \cdot q \equiv -1 \mod M$
Output: $r \in \{0, \ldots, 2q-1\}$ such that $r \equiv x \cdot M^{-1} \mod q$

1: $u \leftarrow (x \cdot q_{inv}) \mod M$ ▷ Reduction explicitly computed
2: $r \leftarrow (x + u \cdot q)/M$ ▷ Exact division ($x + u \cdot q \equiv 0 \mod M$)
3: **return** r

3.5.2 Cortex-M4

The embedded platform targeted by our work is the ARM Cortex-M4. This is the preferred processor for benchmarks of post-quantum algorithms in the framework of the NIST project. It implements the ARMv7E-M architecture offering 16-bit SIMD (Single Instruction Multiple Data) instructions called the DSP extension. It has 14 general purpose registers that are available for the programmer. In practice, we follow the steps of the PQM4 library [68, 70] and work with a STM32F4DISCOVERY board.

3.5.3 SIMD and Useful Instructions

The reason why this platform offers large room for optimization is the availability of parallel instructions used to speed up the NTT. We give here a quick list of useful instructions that are needed to understand the work presented in Section 3.6.1. We use the subscripts h and l to denote the high/low order bits of a register. Every register r is 32 bits long and

its two half-words r_h and r_l are 16 bits long. We use the notation $r = [r_h, r_l]$ to denote that r contains r_h in its high part and r_l in its low part.

Load and stores

Memory access is done through the usage of simple load (`ldr`) and store (`str`) instructions that accept an optional immediate offset in addition to a register containing a pointer. Several variants permit to instead automatically add the offset to the pointer before or after performing the load/store. The two several facts are relevant for us:

- The vanilla `ldr` and `str` instructions load 32 consecutive bits from memory but there exists versions loading only half-words (16 bits) called `ldrh` and `strh`.
- Thanks to pipelining, consecutive loads and stores can offer economies of scale, the first instruction takes 2 cycles as usual and the following ones are performed in a single cycle.

Parallel addition and subtraction

The instructions `usub16` and `uadd16` offer the possibility to interpret the 32-bit registers as vectors of two 16-bit values and add or subtract them in parallel. This is especially helpful since polynomials coefficients in NEWHOPE and KYBER are stored on 16 bits. For example, the naive polynomial addition can thus be performed by chunks of two coefficients.

- `uadd16 c,a,b` $\rightarrow c = [a_h + b_h, a_l + b_l]$
- `usub16 c,a,b` $\rightarrow c = [a_h - b_h, a_l - b_l]$

Signed multiplication on half-words

A very powerful and versatile instruction, or family of instructions is `smul[bt][bt]`. It offers the possibility to execute a $16 \times 16 \rightarrow 32$ signed multiplication. The `b` and `t` part (standing for bottom and top) selects which half-word of each operand is used for the multiplication.

- `smulbb c,a,b` $\rightarrow c = a_l + b_l$
- `smulbt c,a,b` $\rightarrow c = a_l + b_h$
- `smultb c,a,b` $\rightarrow c = a_h + b_l$
- `smultt c,a,b` $\rightarrow c = a_h + b_h$

Those instructions all have a variant that accumulate the result in a specified register called `smla[bt][bt]`. For example:

- `smlabb d,a,b,c` $\rightarrow d = c + a_l * b_l$

Left and right shifts

Naturally, the instruction set also offers simple instructions such as shifts.

- `asr b,a,#n` $\rightarrow b = a >> n$
- `lsr b,a,#n` $\rightarrow b = (unsigned)a >> n$
- `lsl b,a,#n` $\rightarrow b = a << n$

Packing instructions

We will sometime face the difficulty that half-words in different registers have to be reorganized in a single one before using multiple parallel instructions. This is performed with packing instructions.

- `pkhbt c, a, b, lsl#n` $\rightarrow c = [(b << n)_h, a_l]$
- `pkhtb c, a, b, asr#n` $\rightarrow c = [a_h, (b >> n)_l]$

Choosing n in $\{0, 16\}$ effectively offers to put all the possible combinations of half-words of a and b into c.

3.6 Original NewHope NTT

The first contribution is an ARM assembly implementation of the NTT used in NEWHOPE that was pushed to PQM4 in September 2019 [69]. NEWHOPE's NTT closely follows the decimation in frequency method of Section 3.3.2. We give the exact C code of the reference implementation in Listing 3.1 since every detail is important for the assembly version. We note that, as illustrated by Equation 3.3, a pointwise multiplication by powers of γ before the forward transformation and powers of γ^{-1} after the inverse transformation must be performed. Since these are straightforward and pretty fast, they are not discussed here. In NEWHOPE's code, the NTT function of Listing 3.1 is used as a subroutine of two higher level functions: `poly_ntt` and `poly_invntt`. We focus on the dimension 512 NTT, the dimension 1024 case is treated similarly.

```
1  void ntt(uint16_t * a, const uint16_t* omega)
2  {
3    int i, start, j, jTwiddle, distance;
4    uint16_t temp, W;
5
6
7    for(i=0;i<9;i+=2)
8    {
9      // Even level
10     distance = (1<<i);
11     for(start = 0; start < distance;start++)
12     {
13       jTwiddle = 0;
14       for(j=start;j<NEWHOPE_N-1;j+=2*distance)
15       {
16         W = omega[jTwiddle++];
17         temp = a[j];
18         a[j] = (temp + a[j + distance]); // Omit reduction (be lazy)
```

```
19            a[j + distance] = mont_reduce((W * ((uint32_t)temp + 3*NEWHOPE_Q - a[j + distance])));
20          }
21        }
22        if(i+1<9){
23          // Odd level
24          distance <<= 1;
25          for(start = 0; start < distance;start++)
26          {
27            jTwiddle = 0;
28            for(j=start;j<NEWHOPE_N-1;j+=2*distance)
29            {
30              W = omega[jTwiddle++];
31              temp = a[j];
32              a[j] = (temp + a[j + distance]) % NEWHOPE_Q;
33              a[j + distance] = mont_reduce((W * ((uint32_t)temp + 3*NEWHOPE_Q - a[j + distance])));
34            }
35          }
36        }
37      }
38    }
```

Listing 3.1: NTT of NEWHOPE

The NTT has $9 = \log_2 512$ stages (also called layers) performed in pairs by the outer for-loop. Those stages correspond to the deepness of the recursive calls in the FFT methods. They can also be seen (as the columns) in Figure 3.3 even though the NTT depicted there is not the one of NEWHOPE.

Structure of a Layer

Each layer performs a sequence of $256 = \frac{512}{2}$ Gentleman-Sande butterflies. The organization of the butterflies is actually the reverse of the one in Figure 3.3. Let us call distance-n butterfly a butterfly having its inputs n coefficients apart. The first layer performs 256 packs of one distance-1 butterfly, the second layer performs 128 packs of two distance-2 butterflies, etc ... Finally, the last layer performs 2 packs of 128 distance-128 butterflies. The two inner loops correctly organize those $256 = \frac{n}{2}$ butterflies at each stage. Generalizing to an arbitrary dimension n, the full NTT requires the computation of $\frac{n}{2} \cdot \log_2 n$ butterflies and thus has a complexity $\mathcal{O}(n \log n)$.

Gentleman-Sande butterfly

At the heart of the inner loop is the computation of a Gentleman-Sande butterfly. For an appropriate root of unity ω and two values a, b, it computes $a + b$ and $\omega \cdot (a - b)$. In the optimization section, we will call a the first input of the butterfly and b the second input of the butterfly. The difference between odd and even layers is that even layers use lazy reductions in the first part of the butterfly. Indeed, NEWHOPE sets q to $12289 < 2^{14}$ and stores polynomials as arrays of 16-bit unsigned integers, which means that the addition of two reduced values lies in the range $[0, 2q-2]$, which does not overflow. It is thus possible to skip a reduction every other layer. We note that the reduction on odd layers is performed with the % operator in the reference code. It will be replaced by a Barrett reduction in the optimized version. The second part of the butterfly presents more difficulties. First, the inner operation is a subtraction, which could lead to an underflow on unsigned operands. Since any multiple of q can be freely added without changing the result, $3q$ is added to

the first operand such that the result will always be positive ($3q$ is the largest multiple that cannot overflow the addition since intermediate values can grow up to $2q$ because of lazy reductions). Second, the multiplication with the root of unity will give a result on more than 16 bits. Hence, a reduction right after this multiplication has to be performed. Fortunately, since roots are precomputed and stored in Montgomery domain, a cheap Montgomery reduction right after the multiplication will yield the correct result modulo q.

3.6.1 Optimized Assembly Code on Cortex-M4

Compilers are pretty good at outputting fast assembly code from implementations in low-level programming languages such as the reference C code of the NTT used in NEWHOPE. However there are some tasks that are not so easy for the compiler and that can be sped-up using handmade assembly optimizations. This can lead to a significant reduction of computation time. For example, taking the current numbers of PQM4, the reference implementation of KYBER512 takes approximately 900 000 cycles to encapsulate, while the M4 optimized version needs approximately 600 000 cycles. Here, we focus on the following:

- Maximizing parallelism using SIMD instructions;
- Maximizing register usage to reduce memory access.

The state-of-the-art implementation of NEWHOPE on Cortex-M4 before our work was the one presented in [13] in 2016. A more up-to-date work on a NTT algorithm on Cortex-M4 is the implementation of the transform used in KYBER from [31]. The contribution of our work is to implement techniques of [31] in the NTT of NEWHOPE. These changes outperformed the implementation of [13] but were later superseded by the results of Section 3.7.

Montgomery reduction

Algorithm 14 Cortex-M4 assembly Montgomery reduction

Input: registers `q, qinv, tmp, v`

```
1: smulbt tmp, v, qinv
2: smulbb tmp, tmp, q
3: usub16 tmp, v, tmp
4: asr tmp, #16
```

NEWHOPE's reference code implements the Montgomery reduction of Algorithm 13 with $M = 2^{18}$, the choice of a power of two is obvious since the algorithm requires a division by M and a reduction modulo M and 18 is the maximal exponent such that the multiplication $u \cdot q$ does not overflow in a 32-bit data type (because u is a value reduced modulo M and $2^{13} < q < 2^{14}$). In our optimized implementation, we changed the constant to $M = 2^{16}$. This modification improves the performances of the reduction on Cortex-M4 since it offers instructions operating directly on chosen half-words. The reductions modulo M and the division by M can thus be performed implicitly at no

cost at all by choosing the appropriate instruction. For example, let us say we want to compute `c=(unsigned)(a*b)>>16; d=k*c;`, a straightforward translation to assembly would result in 2 multiplications and 1 shift. But if we compute the second multiplication as `smultb d, c, k`, the shift can be omitted since the instruction will directly use the 16 most significant bits of `c` for the multiplication. The Montgomery reduction is the same as the one of [31] and is given in Algorithm 14 and [114]. The slight difference with the usual Montgomery reduction is that the input is multiplied by q^{-1} instead of $-q^{-1}$ at the beginning, which then requires a subtraction instead of an addition. The right shift instruction at the end can be omitted if the next instruction can deal with inputs in MSB position. The first instruction is `smulbt` because the constant q^{-1} is stored in the high part of the register having the alias `qinv`. Actually, in our implementation, both q and q^{-1} are stored in the same register of the processor.

Double butterfly

Algorithm 15 Cortex-M4 assembly Double Butterfly

Input: registers `root, q, qinv, tmp1, tmp2, c_0, c_1`

```
1: usub16 tmp2, c_0, c_1
2: uadd16 c_0, c_0, c_1
3: smulbb tmp1, root, tmp2
4: smulbt tmp2, root, tmp2
5: mont_red q, qinv, c_1, tmp1
6: mont_red q, qinv, tmp1, tmp2
7: pkhbt c_1, c_1, tmp1, lsl#16
```

Since coefficients (and more generally intermediate values during the stages of the NTT) are stored on 16 bits and DSP vectorized instructions interpret the 32-bit registers as two 16-bit values, it is possible to parallelize the computation of butterflies. The goal of a double butterfly is to compute two consecutive butterflies of the same pack using vectorized instructions. The first step is to load, in two registers, the four 16-bit values corresponding to the first and second inputs of each butterfly. Since the two first inputs and the two second inputs are consecutive in memory, this can be efficiently done using the `ldr` instruction loading 32 bits in a register. We load in `c_0` the two first inputs and in `c_1` the two second inputs. We insist on the fact that `c_0` and `c_1` now contain four 16-bit values that can be used to compute two butterflies. Once inputs are loaded, the code of Algorithm 15 is used to compute the two butterflies in parallel. Beside the root, the constants and the inputs, two extra temporary registers are used. Indeed, since the multiplication yields a result on 32 bits before the reduction, all the computations cannot take place in the 16-bit half-words of `c_0` and `c_1`. We remark that, on Cortex-M4, the vectorization is only available for *signed* instructions. This means that a signed version of the reference code had to be written first.

First layer butterflies

While double butterflies were already used in [31], an additional difficulty specific to the NTT of NEWHOPE appeared on the first layer. Indeed, in the first stage, the packs of

butterflies are of size 1 (same as the last stage of Figure 3.3), which means that two consecutive values in memory correspond to the first and second inputs of the *same* butterfly. Hence, a slightly modified algorithm had to be used to compute butterflies of the first layer. We load both inputs of two consecutive butterflies using two `ldr` instructions and perform reorganization in `c_0` and `c_1` with packing instructions to compute two first layer butterflies in parallel. The reason why a code slightly different from Algorithm 15 has to be used is that since both butterflies belong to different pack, the multiplication inside both of them is performed with a different root of unity.

Loops

Loops do not really offer any optimization window. The major issue with them is that the loop counter has to be stored somewhere and will either use a register or incur loads and stores from memory. However, since the NTT instructions does not depend on the inputs, the loops could be fully unrolled and invisible in the assembly code. This would of course notably increase code size. For example, the latest available Cortex-M4 optimized code for NEWHOPE in PQM4 is around 1.5 times larger than the reference code while fully unrolling all the loops of polynomial operations increases code size up to a factor of ten (depending on the parameter set used). There is no right answer for this issue, whether a huge code size is acceptable depends on the context in which it is used. We followed the strategy of previous works by unrolling all the stages and the smaller of the two inner loops.

Layer merging

Beside vectorized instructions, minimizing the number of memory access is very important and can be optimized by writing directly the code in assembly. Where memory access are needed, they should be sequentially organized since isolated loads take 2 cycles while consecutive access benefit from pipelining and reduce the cost of subsequent load/store instructions to a single cycle. In the NTT, it is possible to merge the computation of several layers in order to avoid storing and reloading intermediate values. The goal is to load enough values from memory such that more than one layer can be computed without storing the intermediate output of butterflies in memory. Figure 3.8 illustrates the merge of the two first layers, if only 2 coefficients can be stored in the registers at the same time, the procedure would be:

1. Load $\{p_0, p_1\}$;
2. Compute and store $\{p'_0, p'_1\}$;
3. Load $\{p_2, p_3\}$;
4. Compute $\{p'_2, p'_3\}$ and store p'_3;
5. Load p'_0, compute and store $\{p''_0, p''_2\}$;
6. Load $\{p'_1, p'_3\}$, compute and store $\{p''_1, p''_3\}$.

In total 7 loads and 7 stores are performed. On the other hand, if 4 coefficients can be hold in registers at the same time, we can simply load the 4 values $\{p_0, p_1, p_2, p_4\}$ and directly compute and store the output $\{p''_0, p''_1, p''_2, p''_4\}$ without storing in memory the p'_i. Beside the

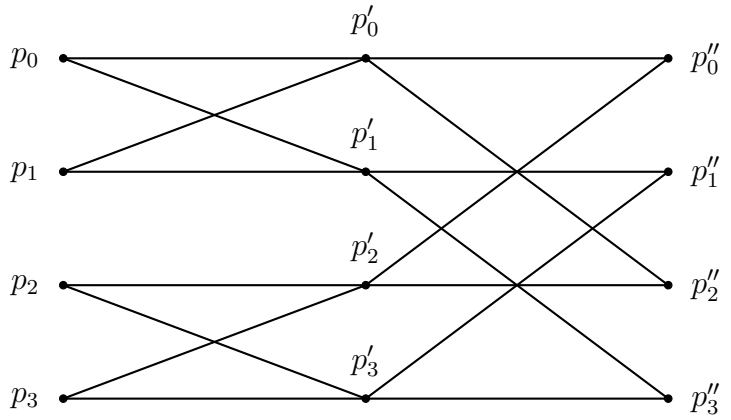

Figure 3.8: Layer Merging. If inputs (p_i) are loaded by chunks of 4, the outputs (p''_i) can be computed without storing the intermediate values (p'_i) in memory.

direct gain in number of instructions, all the loads and all the stores are consecutive, which was not the case in the procedure above. For the sake of simplicity, the explanation above makes the assumption that a butterfly is computed in place from its inputs. Of courses in practice, extra registers are needed for the root of unity, constants and temporary values. Nevertheless, since those extra registers are needed for all butterflies, the assumption can be made without loss of generality. The final point is that the more registers we have, the more layers we can merge and the better the performances are. Unlike [31], we were able to merge the three first layers before merging the others 2 by 2. This is due to the special structure of the first layer of the NTT of NEWHOPE. In full generality n layers can be merged if 2^n values can be loaded at the same time. In our implementation, we have 4 registers available (the allocation of the 14 registers can be found in Table 3.4), so we can merge layers by 2. However, keep in mind we actually load 2 coefficients per register and compute double butterflies. We thus actually have 8 values in the registers at the same time. We could then hope to merge 3 layers. Unfortunately, we do not load 8 *arbitrary* values since a register holds a pair of 2 values that are *consecutive* in memory. In short, we indeed have 8 values at the same time, but not the ones we need to merge 3 layers. The only exception is the first layer. Since it is made of distance-1 butterflies, the 8 values loaded happen to be the ones needed for the merge, and this is why we could perform the three first layers at once.

Performances gain

Since the speed records of this work have been subsequently beaten by the one presented in the next section, we do not detail them much here. The NTT was around 25% faster than the state-of-the-art implementation from [13] and the overall gain for the scheme is presented in Table 3.5. The gain might look slim for a 25% increase in the main operation of the PKE. This symptomatic of a scheme generating a lot of random values from SHAKE256. Actually, as it will be explained in greater details later, optimized schemes can spend up to 80% of their run-time in the hash function. Hence, even if the polynomial multiplication is greatly enhanced, the impact on the whole scheme is less important. However, since the design of the scheme does not depend on the random number generator used, optimizing the NTT is still crucial. Indeed, schemes are mostly designed with an abstract random oracle or extendable-output function used to generate

Register	High part	Description
r0	p	Pointer to the polynomial
r1	root	Pointer to the array of roots of unity
r2	c_0	Coefficient 0
r3	c_1	Coefficient 1
r4	c_2	Coefficient 2
r5	c_3	Coefficient 3
r6	q/qinv	Constants q (low part) and q^{-1} (high part)
r7	root12	Root used in the first layer of each pack of 2 merged layers
r8	root3	Root used in the second layer of each pack of 2 merged layers
r9	loop	loop counter
r10	barrett_c	Constant used for Barrett reductions
r11	tmp1	Temporary value 1
r12	tmp2	Temporary value 2
r14	tmp3	Temporary value 3

Table 3.4: Registers allocation for the NTT of NEWHOPE. In the first three layers, r8 is actually used as an offset needed to compute the address of the root of unity.

Scheme	PQM4 with NTT of [13]	Our work before Section 3.7
NEWHOPE-CPA-512	915293	700489
NEWHOPE-CPA-1024	1495457	1371633
NEWHOPE-CCA-512	1134083	918558
NEWHOPE-CCA-1024	1903231	1777918

Table 3.5: Overall performance gain for NEWHOPE on Cortex-M4 in average number of cycles

random numbers which is latter instanciated with a concrete hash function. Thus changing the random number generator does not change the core design of the scheme. This means that using a faster generator would be possible and would give more weight to the speed of the NTT in the overall computation time.

3.7 Cortex-M4 Optimizations for {R,M}LWE Schemes (CHES 2020)

The second part of this chapter describes the optimization work [6] co-authored with Erdem Alkim, Yusuf Alper Bilgin and Murat Cenk.

Contributions

In the following sections, we describe an optimized Cortex-M4 implementation of NEWHOPE, KYBER, and a recently proposed derivative of those schemes called NEWHOPE-COMPACT. We present various optimizations, mainly in terms of speed and stack usage on the ARM microcontroller. Since those schemes naturally share structural similarities, general improvements are applicable to all of them. Our implementation outperforms the current state-of-the art for KYBER and NEWHOPE and gives a unified

framework to compare the three schemes since they use the same level of optimization, which was not the case in previous works [68,70]. Our contributions are listed as follows:

- We propose 2 cycles modular reduction implementation for Montgomery arithmetic, which translates subtraction to addition to be able to use special instructions.
- We show that small polynomial multiplications can be implemented efficiently using lazy reduction techniques. Hence, we show that an early termination of the NTT with an optimized base multiplication of polynomials of degree greater than 2 can lead to an overall speed improvement.
- We show that even the target architecture has only 14 usable registers, 16 coefficients can be used during butterfly layers. This allowed us to merge up to 4 layers of the NTT and reduce the number of load and store instructions.
- We provide trade-offs between stack usage and speed of the implementation for computing addition of two polynomials after an NTT based polynomial multiplication.

Our code is currently publicly available at

`https://github.com/erdemalkim/NewHope-Compact-M4`

and, at the time of writing, in the PQM4 library.

3.7.1 Polynomial Multiplication in Kyber

Similarly to NEWHOPE, the polynomial multiplication of KYBER is performed in a ring of the form $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$. The main difference is that since we saw in Section 3.4.2 that the parameters used are $q = 3329$ and $n = 256$, a full NTT cannot be performed since $2n$-th roots of unity exist in $\mathbb{Z}_q[X]$ only if $q \equiv 1 \mod 2n$, which is not the case here. Nevertheless, $q \equiv 1 \mod n$ and thus n-th roots of unity do exist. This means that, following the algebraic approach from Section 3.3.2, it is not possible to compute a full NTT

$$\mathbb{Z}_q[X]/\langle X^n + 1\rangle \to \prod_i \mathbb{Z}_q[X]/\langle X - \gamma^i\rangle,$$

but it is possible to compute a "partial" NTT

$$\mathbb{Z}_q[X]/\langle X^n + 1\rangle \to \prod_i \mathbb{Z}_q[X]/\langle X^2 - \gamma^i\rangle.$$

The partial NTT is a fairly standard FFT using Cooley-Tukey butterflies in the forward transform and Gentleman-Sande butterflies in the inverse mapping. The only major difference is that it stops a layer before the end. Since it is a variant of the FFT techniques described above, we omit the details here. This approach has the two following consequences:

- The computation of the NTT is slightly faster since it skips the last layer;
- The base multiplication $\circ$ is a multiplication in short rings of the form $\mathbb{Z}_q[X]/\langle X^2 - r_i\rangle$ instead of pointwise multiplications.

The base multiplication is thus more expensive while the NTT is less expensive. At the end of the day, the computational cost of the multiplication in $\mathbb{Z}[X]_q/\langle X^n + 1\rangle$ is similar to a multiplication using a full NTT. The designers of the scheme thus decided to keep this technique as it offers more flexibility in the parameters choice.

Table 3.6: Parameters of NEWHOPE-COMPACT512, NEWHOPE-COMPACT768 and NEWHOPE-COMPACT1024 derived high level properties [8]

Parameter Set	NH-COMPACT512	NH-COMPACT768	NH-COMPACT1024
Dimension n	512	768	1024
Modulus q	3329	3457	3329
Noise Parameter k	2	2	2
Decryption error probability	2^{-256}	2^{-170}	2^{-181}
Claimed post-quantum bit security	100	163	230
NIST Security Strength Category	1	3	5

3.7.2 NewHope-Compact

Having this new technique in mind, it is natural to wonder if it could be applied to NEWHOPE as well. Indeed, the modulus of NEWHOPE was chosen as the smallest prime congruent to 1 modulo 2048 but nothing prevents reducing the modulus in the same way KYBER did. This was proposed at Latincrypt 2019 by Almik, Bilgin and Cenk [8], who are also the co-authors of the contribution we discuss here. The resulting scheme, called NEWHOPE-COMPACT, which is presented as a RLWE KEM based on NEWHOPE and KYBER, is basically NEWHOPE instanciated with the parameters of Table 3.6, that is to say with the modulus and error distribution of KYBER. They also propose a reordering of coefficients to use the same NTT code across parameters. We remark that the authors propose an instanciation of the scheme with dimension $n = 768$, which provides a trade-off between the very compact dimension 512 and the very secure dimension 1024. Since 768 is not a power of two, the corresponding cyclotomic polynomial is not of the form $X^n + 1$ but is actually $X^{768} - X^{384} + 1$. Thus, the NTT strategy described earlier in this chapter do not work. However, [85] showed that a really close variant offering similar performances is possible. Since the mapping is slightly different, the modulus for dimension 768 has been adapted for the underlying field to offer the needed roots of unity.

3.7.3 Speed Optimizations of Polynomial Multiplication

We first describe speed optimizations for the multiplication in $\mathcal{R}_q$. Such a multiplication is composed of the computation of the NTT, its inverse, and a base multiplication between elements in NTT domain. The base multiplication operation depends on the targeted algorithm. For NEWHOPE, it is simply the usual multiplication in $\mathbb{Z}_q$ while for the others algorithms, KYBER and NEWHOPE-COMPACT, it is a small polynomial multiplication modulo $X^{n'} + r$ performed with a schoolbook algorithm. KYBER requires a base multiplication with $n' = 2$ for all parameter sets while NEWHOPE-COMPACT uses $n' = n/128$. The optimizations we describe now are applied to KYBER, NEWHOPE and NEWHOPE-COMPACT. Their exact implementation might slightly differ from one algorithm to the other but we sometime gloss over differences to avoid overloading the description.

Usage of Montgomery, Barret and Lazy reductions

The Montgomery reduction of [31] takes 3 clock cycles. This is actually the one we presented in Algorithm 14 except that the last shift is not performed and the output is instead defined as the high part of the output register. In our work, we decided to store $-q$ instead of q and thus to have a Montgomery reduction performing an addition instead of a subtraction as last instruction. This way, we are able to perform the multiplication and the addition in a single clock cycle using the multiply and accumulate instruction `smlabb`, which saves one cycle per reduction (see Algorithm 16). Since Montgomery reductions are used in each butterfly of each layer during a NTT, this saves a sensible amount of cycles. In KYBER, there are 896 reductions in NTT/INTT and 512 reductions in base multiplication, this means that 3200 clock cycles are saved for a full multiplication in $\mathcal{R}_q$.

Algorithm 16 Signed Montgomery reduction using Montgomery factor $\beta = 2^{16}$.

Input: a where $-\frac{\beta}{2}q \leq a < \frac{\beta}{2}q$
Output: reduced $a \rightarrow r'$ where $r' = \beta^{-1}a \pmod q$, and $-q < r' < q$

```
1: smulbb t, a, mqinv        ▷ t ← (a mod β) · (−q^{-1})
2: smlabb a, t, q, a         ▷ a_top ← ⌊((t mod β)·q + a) / 2^16⌋
```

Similarly to [31], we use the Barret reduction of [114]. The assembly implementation of this reduction on a packed argument, that is to say taking as inputs two 16-bit coefficients packed in a single 32-bit register, is given in Algorithm 17 and takes 8 clock cycles. Actually, the two cycles Montgomery reduction that we just presented enables a faster algorithm to reduce packed values in normal domain (Algorithm 18). Indeed, 2 cycles to convert each value to Montgomery domain, 4 cycles to reduce both values and 1 cycle to repack them correctly means a reduction on packed argument in 7 cycles, which beats the Barrett reduction, even for values not in Montgomery domain. The only small drawback is that its output range is larger. Thus, when it does not matter, we always use Montgomery reductions.

The usage of lazy reductions in the NTT depends on the type of butterfly used and the size of the modulus. We did not propose any improvement on this side in the NTT but made a larger use of lazy reductions in the base multiplication algorithm. Instead of reducing after each multiplication in the schoolbook algorithm, we reduce only after a sum of products of coefficients. For example, in the base multiplication of KYBER given in Algorithm 19, we compute c_0 as

$$c_0 \leftarrow (((a_0 \cdot b_0 + a_1 \cdot b_1) \bmod q) \cdot r) \bmod q$$

instead of reducing after each multiplication. While the gain is pretty slim yet non negligible for KYBER, it is especially relevant for NEWHOPE-COMPACT that uses a base multiplication on polynomials of larger size. In particular, for NEWHOPE-COMPACT1024, this is a multiplication in $\mathbb{Z}_q[X]/(X^8 - r)$ and we can add up to 8 products before reducing. However, using lazy reductions, one should always be careful to not overflow. In our case, working in absolute value, each coefficient is < 3329 and thus, a sum of 8 products is $\leqslant 8 * 3328^2 = 88604672$ which is below the maximum value $2^{15} \cdot 3329 = 109084672$ that the Montgomery reduction can handle.

Algorithm 17 Barrett reduction on packed argument using $\beta = 2^{16}$.

Input: 32-bit signed integer a with a_h and a_l both containing coefficients, precomputed Barrett constant v

Output: r = $r_h \mid r_l$ where $r_h \equiv a_h \pmod q$, $r_l \equiv a_l \pmod q$, $0 \leq r_h, r_l \leq q$ and $0 \leq q < 2^{15}$

1:	`smulbb t1, a, v`	$\triangleright\ t_1 \leftarrow a_l \cdot v$
2:	`smultb t2, a, v`	$\triangleright\ t_2 \leftarrow a_h \cdot v$
3:	`asr t1, t1, #(log(β) + ⌊log(q)⌋ − 1)`	$\triangleright\ t_1 \leftarrow t_1 >> (\log(\beta) + \lfloor\log(q)\rfloor - 1)$
4:	`asr t2, t2, #(log(β) + ⌊log(q)⌋ − 1)`	$\triangleright\ t_2 \leftarrow t_2 >> (\log(\beta) + \lfloor\log(q)\rfloor - 1)$
5:	`smulbb t1, t1, q`	$\triangleright\ t_1 \leftarrow t_1 \cdot q$
6:	`smulbb t2, t2, q`	$\triangleright\ t_2 \leftarrow t_2 \cdot q$
7:	`pkhbt t, t1, t2, lsl#16`	$\triangleright\ t \leftarrow (t_1 \& 0xFFFFu) \mid (t_2 << 16)$
8:	`usub16 r, a, t`	$\triangleright\ r_h \leftarrow a_h - t_h$ and $r_l \leftarrow a_l - t_l$

Algorithm 18 Signed Montgomery reduction on packed argument using Montgomery factor $\beta = 2^{16}$.

Input: 32-bit signed integer a with a_h and a_l both containing coefficients, precomputed Montgomery constant v

Output: $r = r_h \mid r_l$ where $r_h \equiv a_h \pmod q$, $r_l \equiv a_l \pmod q$

1:	`smulbb t1, a, v`	
2:	`smulbb r1, t1, mqinv`	$\triangleright\ r_1 \leftarrow (t_1 \mod \beta) \cdot (-q^{-1})$
3:	`smlabb r1, r1, q, t1`	$\triangleright\ r_{1_h} \leftarrow \left\lfloor \frac{(r_1 \mod \beta) \cdot q + t_1}{2^{16}} \right\rfloor$
4:	`smultb t2, a, v`	
5:	`smulbb r2, t2, mqinv`	$\triangleright\ r_2 \leftarrow (t_2 \mod \beta) \cdot (-q^{-1})$
6:	`smlabb r2, r2, q, t2`	$\triangleright\ r_{2_h} \leftarrow \left\lfloor \frac{(r_2 \mod \beta) \cdot q + t_2}{2^{16}} \right\rfloor$
7:	`pkhtb r, r2, r1, asr #16`	$\triangleright\ r \leftarrow (r_{2_h} \mid (r_{1_h} >> 16))$

Algorithm 19 Multiplication of polynomials in $\mathbb{Z}_q[X]/(X^2 - r)$ for KYBER.

Input: a and $b \in \mathbb{Z}_q[X]/(X^2 - r)$ where r is a root of unity.

Output: $c \in \mathbb{Z}_q[X]/(X^2 - r)$.

1: **function** basemul(a, b)
2: $\quad c_0 \leftarrow (a_0 \cdot b_0) \bmod q + ((a_1 \cdot b_1) \bmod q) \cdot r) \bmod q$
3: $\quad c_1 \leftarrow (a_0 \cdot b_1) \bmod q + (a_1 \cdot b_0) \bmod q$
4: $\quad$ **return** c
5: **end function**

More Aggressive Layer Merging

As previous works remarked, reducing the number of loads and stores by merging several layers of the NTT can lead to noticeable performance gain. While [31] merged layers two by two using 4 registers to store 8 coefficients and kept other registers to store constants and loops counters, we used a more aggressive strategy by storing 16 coefficients in 8 registers. This enables the possibility to merge three and sometimes four (same situation as explained earlier in the layer merging of NEWHOPE) layers. Naturally, sacrificing some registers to store more coefficients comes at a certain cost since they were used to keep constants, loop counters and pointers. Since the NTT always performs the same amount of computations, we can unroll all the loops using the `.rept` directive and get rid of loop counters but, of course, doing so dramatically increases code size and thus, we decided to propose it only as an option. Therefore, we instead decided to rely more actively on register spilling, that is to stay store and load seldom used values on the stack instead of keeping them always in registers, and constant reloading from the code. Our conclusion is that, speed-wise, the advantages of merging more layers outweigh the disadvantages of refilling the registers with appropriate values when needed.

3.7.4 Optimization of NewHope and NewHope-Compact for Stack Usage

On embedded devices, it is often the case that memory usage introduces a significant bottleneck. Outside of real-time systems, one can always wait for a slow algorithm, but if the algorithm needs more memory than what can be found on the device, it is completely unusable. While the Cortex-M4 on our board offers quite a large amount of memory, we decided to optimize for stack usage as well.

The approach we took was to reduce the minimum amount of stack space required to compute the cipher while keeping performance mostly unaffected. While it is possible to follow a more aggressive approach to reduce stack usage, such implementations would be considerably slower than ours. The three main metrics regarding implementation are speed, stack usage and code size. The one to optimize severely depends on the context in which the cipher will be used. In our work, we tried to optimize the first two while keeping the last one reasonable.

Key generation

The core of the key generation is the computation of $\hat{b} \leftarrow \hat{a} \circ \mathsf{NTT}(s) + \mathsf{NTT}(e)$. Since each coefficient of the output of the NTT depends on *all* the coefficients of the input, all the coefficients of s and e must have been generated before proceeding to the addition. Hence, at least two full polynomials should be stored in memory. To reduce the memory usage, we used the observation that polynomial multiplication can be performed on-the-fly in the NTT domain and, likewise, adding an error to a polynomial can be performed on-the-fly in *normal* domain. Indeed, the operation $\circ$ works sequentially on parts of its inputs (one coefficient at the time for NEWHOPE and four, six or eight for NEWHOPE-COMPACT depending on the parameter set used) and does not need all of them in memory at the same time. Similarly, the error polynomial can be computed coefficient by coefficient and added directly, but only if the addition considered is in the normal domain. This

is why instead of computing

$$\hat{b} \leftarrow \hat{a} \circ \mathsf{NTT}(s) + \mathsf{NTT}(e),$$

we compute

$$\hat{b} \leftarrow \mathsf{NTT}(\mathsf{INTT}(\hat{a} \circ \mathsf{NTT}(s)) + e)$$

and perform the multiplication and the addition on-the-fly. This requires one more INTT but allows to only store *one* polynomial in memory, containing s and $\hat{b}$ subsequently. This approach reduces stack usage significantly and since our benchmarks show that computing one extra optimized INTT only increase the key generation time by around 5%, we believe that this is a good trade-off. This trick can be similarly applied to KYBER. Note that the small *relative* cost of this technique is specific to our context and is mainly due to the fact that hashing is the main performance bottleneck. This issue will be discussed in more detail in Section 3.7.6. If a faster hash function is used, the decrease in performance will be higher than 5%. That being said, the *absolute* cost of the trick is always the same and is one INTT.

Encryption

The encryption procedure is mainly driven by the following computations:

1. $\hat{t} \leftarrow \mathsf{NTT}(s')$
2. $\hat{u} \leftarrow \hat{a} \circ \hat{t} + \mathsf{NTT}(e')$
3. $v' \leftarrow \mathsf{INTT}(\mathsf{DecodePoly}(\hat{b'}) \circ \hat{t}) + e'' + \mathsf{Encode}(\mu)$

The first two yield a situation similar to the key generation but unfortunately require two polynomials on the stack frame. Indeed, since $\hat{t}$ appears in the second and last computation, the result of $\hat{a} \circ \hat{t}$ *cannot* be stored in the same memory space as $\hat{t}$ (and since it would need to go through a INTT, it does need to be fully stored). Once $\hat{u}$ is computed, it can be packed in the ciphertext and free one of the two polynomials. The last computation is quite friendly for stack usage. Since both the base multiplication and the addition operate on small portions of the polynomial, $e'' + \mathsf{Encode}(\mu)$ can be computed coefficient by coefficient and $\hat{b}$ can be partially unpacked from the inputs, it could technically be computed with one polynomial plus a small overhead in the stack frame. Since two polynomials were already allocated previously and only maximal stack usage is relevant, we actually fully unpack $\hat{b}$. Finally, the stack usage is bigger than the one of the key generation because of the extra polynomial stored.

Decryption

The decryption of NEWHOPE is quite lightweight in terms of stack usage. However, since the CCA transform is running the encryption procedure during decryption, the stack usage is essentially the same as for encryption.

3.7.5 Tradeoffs Between Secret Key Size and Speed

There are different possible tradeoffs between secret key size and speed of the implementation [29, 113]. For example, if the secret key size is critical, one can only store the

seed used for all randomness and perform key generation again during decapsulation. As also stated by [29, 113], another optimization could to store the secret key in the normal domain instead of the NTT domain because then, each coefficients could be compressed to 3 bits (since their possible values are between -2 and 2). Since our implementations of the NTTs used in KYBER and NEWHOPE-COMPACT are fast, we decided that such modifications are good trade-offs. Even though it is usually stated that the most time consuming part of such algorithms is the randomness generation and hashing, we observed that sampling polynomials from the secret distribution is fast enough due to the low entropy of each coefficient. Sampling the secret key is actually lightweight in comparison to the generation of the public parameter a because we can extract two coefficients from only one byte when sampling from the centered binomial distribution while uniform sampling needs two bytes to extract only one coefficient. Hence, we decided to store only the seed, whose size is 32 bytes, to sample the secret key. Then, the secret key is sampled again during decapsulation, and is transformed to NTT domain. These operations reduce the secret key size by 736 bytes for KYBER512 and NEWHOPE-COMPACT512, 1120 bytes for KYBER768 and NEWHOPE-COMPACT768, and 1504 bytes for KYBER1024 and NEWHOPE-COMPACT1024. On the other hand, they increase the decapsulation time by around 7% for KYBER and 9% for NEWHOPE-COMPACT.

3.7.6 Results and Comparison

Our optimizations were implemented in the three siblings schemes NEWHOPE, NEWHOPE-COMPACT, and KYBER. Comparing different schemes across parameter sets is often complicated because performance is always strongly correlated with the targeted security level. Most of the implemented schemes propose parameter sets for NIST security levels 1, 3, and 5, which correspond to 128, 192, and 256 bits of security. Fortunately, since all the schemes involved in our tests are similar and based on {R,M}LWE, the dimension of the underlying lattice problem can currently be roughly translated into NIST security levels. Hence, we compare them for dimensions 512, 768 (if available), and 1024, which correspond to the three aforementioned security levels.

3.7.7 Speed Comparison

The results of our benchmarks in terms of speed can be found in Table 3.7. The code was compiled and run in the same conditions as the schemes benchmarked in `pqm4` [68]. We use the latest `arm-none-eabi-gcc` release (version 9.2.1) that is currently available. We compare the two candidates NEWHOPE and KYBER against their previous Cortex-M4 optimized implementations available in `pqm4` and also add the newcomer NEWHOPE-COMPACT. One can see that NEWHOPE and KYBER perform around 10% better with our optimizations, while using less stack space (see Table 3.8). Furthermore, NEWHOPE-COMPACT is more than 40% faster compared to NEWHOPE and more than 25% faster compared to KYBER for all security levels. This is explained by the two following observations:

- NEWHOPE-COMPACT is a variant of NEWHOPE using a smaller modulus and distribution, which means an increased performance during polynomial multiplication because of lazy-reductions and less hashing needed to sample from the error distribution.

- NEWHOPE-COMPACT is based on RLWE, while KYBER is based on MLWE. Hence, even though they share similar parameter sets, the inherent performance penalty of using the less structured version of LWE hurts KYBER.

Scheme		Previous work	This work Speed	This work Stack Opt.	This work Short sk
NEWHOPE	512	**G**: $588k$ [a] **E**: $918k$ [a] **D**: $904k$ [a]	**G**: $561k$ (-4.6%) **E**: $865k$ (-5.8%) **D**: $820k$ (-9.3%)	**G**: $578k$ (-1.7%) **E**: $865k$ (-5.8%) **D**: $820k$ (-9.3%)	**G**: $555k$ (-5.6%) **E**: $865k$ (-5.8%) **D**: $968k$ $(+7.1\%)$
	1024	**G**: $1\,161k$ [a] **E**: $1\,777k$ [a] **D**: $1\,760k$ [a]	**G**: $1\,117k$ (-3.8%) **E**: $1\,687k$ (-5.1%) **D**: $1\,612k$ (-8.4%)	**G**: $1\,157k$ (-0.3%) **E**: $1\,689k$ (-5.0%) **D**: $1\,614k$ (-8.3%)	**G**: $1\,106k$ (-4.7%) **E**: $1\,686k$ (-5.1%) **D**: $1\,918k$ $(+9.0\%)$
NH-CMPCT	512	-	**G**: $335k$ **E**: $531k$ **D**: $484k$	**G**: $349k$ **E**: $532k$ **D**: $484k$	**G**: $330k$ **E**: $531k$ **D**: $526k$
	768	-	**G**: $501k$ **E**: $782k$ **D**: $717k$	**G**: $524k$ **E**: $784k$ **D**: $718k$	**G**: $494k$ **E**: $782k$ **D**: $786k$
	1024	-	**G**: $658k$ **E**: $1\,022k$ **D**: $940k$	**G**: $686k$ **E**: $1\,025k$ **D**: $941k$	**G**: $648k$ **E**: $1\,022k$ **D**: $1\,030k$
KYBER	512	**G**: $514k$ [b] **E**: $652k$ [b] **D**: $621k$ [b]	**G**: $452k$ (-12.1%) **E**: $586k$ (-10.1%) **D**: $542k$ (-12.7%)	**G**: $461k$ (-10.3%) **E**: $586k$ (-10.1%) **D**: $543k$ (-12.6%)	**G**: $446k$ (-13.2%) **E**: $586k$ (-10.1%) **D**: $579k$ (-6.9%)
	768	**G**: $976k$ [b] **E**: $1\,146k$ [b] **D**: $1\,094k$ [b]	**G**: $860k$ (-11.9%) **E**: $1\,031k$ (-10.0%) **D**: $967k$ (-11.6%)	**G**: $872k$ (-10.7%) **E**: $1\,030k$ (-10.1%) **D**: $966k$ (-11.7%)	**G**: $850k$ (-12.9%) **E**: $1\,030k$ (-10.1%) **D**: $1\,021k$ (-6.7%)
	1024	**G**: $1\,575k$ [b] **E**: $1\,779k$ [b] **D**: $1\,709k$ [b]	**G**: $1\,394k$ (-11.5%) **E**: $1\,603k$ (-9.9%) **D**: $1\,522k$ (-10.9%)	**G**: $1\,410k$ (-10.5%) **E**: $1\,603k$ (-9.9%) **D**: $1\,523k$ (-10.9%)	**G**: $1\,381k$ (-12.3%) **E**: $1\,603k$ (-9.9%) **D**: $1\,595k$ (-6.7%)

[a] `https://github.com/mupq/pqm4/`, commit `be0c421aaecaad4443071bfcf62ad397d4f40832`.

[b] [31]

Table 3.7: Cycle count comparison for the {R,M}LWE schemes improved by our work. **G**: key generation, **E**: encapsulation, **D**: decapsulation.

Moreover, some design decisions affect the performance or the stack usage of the scheme. These design decisions are unrolling NTT loops by using the `.rept` directive instead of the loop counter, optimizing the stack usage, and the trade-off between the size of the secret key and performance. These three decisions can be easily enabled or disabled. The effects of the last two choices on the performance and the stack usage are given in Table 3.7. Using the `.rept` directive instead of the loop counter decreases the cycle count by n per NTT call, where n is the degree of the polynomial for selected parameters. However, the code size increases by a factor of 10 to 50. Optimizing the stack usage decreases the memory used by key generation while increasing the cycle count of the same function. Finally, applying the method described in Section 3.7.5 to reduce the size of the secret key increases the cycle count of the decapsulation while decreasing it for the key generation.

3.7.8 Dominance of Hashing

The speed difference shown in Table 3.7 might look slim at first sight. This is due to the fact that, as pointed out by previous works, those schemes have been optimized so much

Scheme	NEWHOPE (This work)	NEWHOPE [68][a]	NH-CMPCT (This work)	KYBER (This work)	KYBER [31]
512	**G**: 2 056 **E**: 2 864 **D**: 2 880	**G**: 5 960 **E**: 9 168 **D**: 10 296	**G**: 2 160 **E**: 2 984 **D**: 2 984	**G**: 2 392 **E**: 2 344 **D**: 2 360	**G**: 2 952 **E**: 2 552 **D**: 2 560
768	-	-	**G**: 2 600 **E**: 3 936 **D**: 3 936	**G**: 3 240 **E**: 2 856 **D**: 2 864	**G**: 3 848 **E**: 3 128 **D**: 3 072
1024	**G**: 3 072 **E**: 4 904 **D**: 4 920	**G**: 11 080 **E**: 17 360 **D**: 19 576	**G**: 3 176 **E**: 5 024 **D**: 5 024	**G**: 3 776 **E**: 3 744 **D**: 3 760	**G**: 4 360 **E**: 3 584 **D**: 3 592

[a] `https://github.com/mupq/pqm4/`, commit `be0c421aaecaad4443071bfcf62ad397d4f40832`.

Table 3.8: Stack usage comparison for the {R,M}LWE schemes improved by our work. **G**: key generation, **E**: encapsulation, **D**: decapsulation.

that the bottleneck is now the generation of random numbers through hashing instead of the polynomial multiplication procedure. Table 3.9 shows the time spent hashing for all algorithms and parameter sets. As we can see, with a minimum of 66% for the decapsulation of NEWHOPE-COMPACT, all the algorithms are severely dominated by hashing. Even if polynomial multiplications were somehow instantaneous, the results of Table 3.7 would be somewhat similar.

Scheme	Dimension 512	Dimension 768	Dimension 1024
NEWHOPE	**G**: 75% **E**: 80% **D**: 72%	-	**G**: 73% **E**: 78% **D**: 71%
NEWHOPE-COMPACT	**G**: 75% **E**: 78% **D**: 67%	**G**: 72% **E**: 77% **D**: 66%	**G**: 73% **E**: 77% **D**: 66%
KYBER	**G**: 76% **E**: 80% **D**: 69%	**G**: 77% **E**: 80% **D**: 72%	**G**: 78% **E**: 80% **D**: 73%

Table 3.9: Time spent hashing. **G**: key generation, **E**: encapsulation, **D**: decapsulation.

3.7.9 Comparing Polynomial Multiplications

The reader might wonder why to bother optimizing polynomial multiplications further if it is not the bottleneck anymore. The reason is twofold: first, SHAKE is used to expand the seed in every scheme implemented. However, the choice of the seed expansion algorithm is somewhat orthogonal to the scheme and does not affect post-quantum assumptions. Hence, using a faster hash function would reduce the impact of hashing on the performance. Furthermore, it might be unnecessary to use a cryptographic hash function to generate the public parameters. For instance, [30] uses a faster, non-cryptographic RNG to speed-up a scheme based on LWE. Second, even if SHAKE is used, since its

usage will likely grow in all future cryptographic applications, we might eventually see hardware accelerations for it on a lot of architectures. This would naturally drastically decrease the time spent hashing in our schemes and make the polynomial multiplication is the most important optimization target again. Recall that, as stated in Section 3.7.4, this would increase the relative cost of the reduced stack usage trick used in the key generation. Nevertheless, we think that outside of unrealistically fast polynomial generation, the trade-off can still be useful.

Since our work is the first Cortex-M4 implementation of NEWHOPE-COMPACT, we do not have any point of comparison for our technique for this scheme. Table 3.10 shows the speed-up for the dimension of the NTT used in all parameter sets of NEWHOPE and KYBER and the cycle count of all subroutines of the polynomial multiplication for each algorithm and dimension. The total cost of multiplication operations for each scheme is presented in Table 3.11. This table was obtained by summing all the time spent in the three multiplication subroutines: NTT, INTT, and $\circ$. Note that the stack optimized version of our implementation is used in this table to show its actual impact on the performance. It can be seen that KYBER and NEWHOPE-COMPACT have similar performance, while NEWHOPE is slower. This is mainly due to the extra layers of the NTT and the increased number of reductions caused by the larger modulus. Note that our INTT cycles do not include any bitreversal operation, because we need INTT to output a bitreversed order for the stack optimization (Section 3.7.4). To be able to verify test vectors with the reference implementation of NEWHOPE, we have implemented a separate bitreversal operation that takes roughly $4n$-cycle for the selected parameter set, which is not included in the INTT. It can be seen that our implementation of NEWHOPE NTT is slightly slower compared to the implementation from [68] [4], while we have noticeably better performance for the INTT. Table 3.11 shows that even though we have a slower NTT, the total number of cycles spent in polynomial operations is reduced compared to [68].

Scheme	Dimension	NTT	INTT	$\circ$
NEWHOPE	512	28662 (-3.7%)	22836 (-35.8%)	4736 (-13.2%)
	512 ([68][a])	29767	35813	5459
	1024	63387 (+6.1%)	49880 (-30.7%)	9396 (-13.3%)
	1024 ([68][a])	59752	71942	10836
	1024 ([13])	86769	97340	14977
NEWHOPE-COMPACT	512	12799	13052	7052
	768	19647	21226	12749
	1024	25536	26039	18510
KYBER	256	6847 (-11.7%)	6975 (-25.6%)	2317 (-24.7%)
	256 ([31])	7754	9377	3076

[a] `https://github.com/mupq/pqm4/`, commit `be0c421aaecaad4443071bfcf62ad397d4f40832`.

Table 3.10: Comparison of the polynomial multiplication functions of all the schemes. Kyber actually uses the exact same NTT code for all dimensions.

3.8 Conclusion

In this chapter, we described two interleaved implementation works that led to new speed records for some RLWE/MLWE KEM and a paper about optimizations and trade-offs [6].

[4] which is actually our implementation of Section 3.6.1 as it was integrated to PQM4.

Scheme	Dimension	Keygen	Encaps	Decaps
NEWHOPE	512	84896	89632	117204
	512 ([68][a])	64993	106265	147537
	1024	186050	195446	254722
	1024 ([68][a])	130340	213118	295896
NEWHOPE-COMPACT	512	45702	52754	72858
	768	73269	86018	119993
	1024	95621	114131	158680
KYBER	512	50606	48521	73824
	512 ([31])	43320	62095	93132
	768	82860	76245	110712
	768 ([31])	74208	97682	139549
	1024	119748	108603	152234
	1024 ([31])	111248	139421	192118

[a] `https://github.com/mupq/pqm4/`, commit `be0c421aaecaad4443071bfcf62ad397d4f40832`.

Table 3.11: Total time spent in polynomial multiplication subroutines (NTT, INTT, and $\circ$).

The targeted platform is the Cortex-M4 processor which is the embedded system used in the framework of the NIST project. Improvements came from better use of SIMD instructions offered by the ARMv7E-M architecture. More specifically, we proposed new efficient implementations for NEWHOPE, NEWHOPE-COMPACT and KYBER. The core speed optimizations are due to a more aggressive layer-merging strategy and improved reductions in the NTT and the base multiplication. We provided an implementation that has already been integrated into the PQM4 library as well as experiment results comparing the proposed trade-offs. The code is written in a modular fashion that allows the user to easily switch between versions. Our results show that all optimization techniques have advantages and disadvantages and might be useful for different applications.

3.9 Thoughts and Future Works

While I guess there will probably always be some percentages to gain here and there, it is clear that the huge cost of randomness generation is hindering further progress in terms of speed. Even though our improvements on the NTT (which is the core operation) are quite good, only slim gains are perceptible on the whole scheme. Of course, as discussed at the end of the chapter, if future devices are equipped with special SHAKE instructions, the relative weight of those gains will increase, but the inertia of the industry on this matter is hard to predict. Actually, I would argue for benchmarks without the hash function in optimization papers to get a better idea of the improvements. That being said, the huge amount of randomness needed is not so often discussed when comparing post-quantum schemes while, in practice, it might be a clear drawback. Optimization works discussing the problem with randomness generation are good remainders that this issue is crucial to cryptography. However, I think that when we do a comparison between optimizations, this parameter can disappear. This work also shows another problem of optimized implementations: several metrics. To get the absolute fastest code, one can inline and unroll everything and write a single large blob of assembly code; to get the smallest stack usage,

one can keep every polynomial as a seed and generate each coefficient one the fly when needed. Of course, these extreme examples are nonsense but they illustrate the fact that some reasonable choices have to be made. The problem is that the definition of reasonable differs from one person to the other. Naturally, some objectives improvements are possible if they improve one metric while keeping others mostly untouched but my point is that what is considered a better implementation depends a lot on the context. Actually, I think improvements of practical performances are highly tied to design parameters choices. For example, NEWHOPE-COMPACT is largely more efficient than NEWHOPE while the underlying algorithm is basically the same. If the NEWHOPE team decides to support the parameter sets of the compact version, it could give a large boost to the practical side of the submission. Actually with the current state of randomness generation, such scheme should aim to reduce the modulus and the support of the distribution as much as possible to increase speed. However, it is known that too narrow distributions enable some cryptanalysis, so things are not so easy. Regarding future works, I would say that, for the reason exposed above, there is not much room for improvements in the current framework (Cortex-M4 and software SHAKE). Nevertheless, it is important to give the same level of attention to all post-quantum candidates to have fair comparisons in PQM4. I also think that targeting various embedded architectures, from small 8-bit microcontrollers to powerful smartphones, is interesting. Since the groundwork for optimized implementation is already there, porting on other devices should be relatively easy. Finally, if we are talking about embedded devices, side-channel resistant implementations would be welcomed. Not so much countermeasures have been studied yet but, fortunately, this is the topic of the next chapter!

Chapter 4

Protecting Lattice-based Cryptography Against Physical Attacks

4.1 Preamble

This chapter is about side-channel attacks and countermeasures. I was introduced to this topic by the researchers working in my group at the beginning of my PhD. At the time, most of them were working on side-channel related topics. While they were working on symmetric algorithms and I was already taking the path of lattice-based cryptography, I still gained some useful knowledge on the topic. This led to a small research paper with my colleagues from ULB at SPACE 2016 [53]. While its topic is not related to lattice-based cryptography, it will be used to introduce side-channel attacks in this Chapter and a more important research paper discussing the masking countermeasure on the qTESLA lattice-based signature authored with Mélissa Rossi from ENS, published at CARDIS 2019 [61].

4.2 Introduction

In Chapter 3, our goal was to implement as efficiently as possible existing cryptographic algorithms on a specific device. The underlying assumption was that the implemented scheme features all the required security properties and that once the code exactly implements the mathematical operations specified by the designers, the work is done and the user can safely use it. Unfortunately, some unforeseen threats while using the implementation might appear. For example, imagine the user is using a program encrypting some plaintext on its personal computer. What if another program finds an exploit in the operating system and reads some intermediate values in RAM during the encryption procedure? The cipher was originally not designed to be computationaly secure while revealing its computation steps and this could lead to a total reveal of the plaintext and/or the key. Such information is called a *side-channel information*, it was obtained from outside of the usual communication channel of the scheme. Now, it is accepted that meeting security goals on an infected computer is pretty much impossible. Furthermore, it is not "cryptographically" possible to contain every threats of the real world. For example, what if a user writes the secret key on a piece of paper in sight of the attacker? However, there exists plenty of side-channel attacks that can be mitigated in practice that are studied in the literature. Among them, we will focus on the popular power analysis attacks. In those, the

adversary is given the power consumption of an electronic device while it is computing some cryptographic algorithm and tries to derive some secret information from it. This new threat appeared in the literature a bit before 2000 in [73]. A very interesting property it has is that it falls in the category of *non-invasive* attacks. This means that the attacker does not have to modify the device or access its internal components and can perform the attack from the outside. It does not really make it a remote attack since accurately measuring consumption requires to be close (and quite often unrealistically close in academic settings !) but, since the device can be left untouched, the attack is harder to detect.

4.3 Power analysis attacks

A measurement of power consumption over a certain period of time is called a trace. In its simplest version, a power attack will directly derive information from a trace. For example, let us assume the existence of an unrealistic device loading data bit by bit from memory, one per cycle, having its power consumption drastically different when loading a 0 or a 1. Also assume that a measurement device capable to measure consumption at the same frequency as the clock of the device is available. Once the attacker finds the instant in time when the key is loaded to start performing cryptographic operations, just looking at the values in the trace directly reveals it. This type of attack is called a *simple power analysis* (SPA). However, these are quite hard to conduct in practice since the situation above was idealized for the attacker. In real-life, measurements are noisy and the power consumption of individual bits is unlikely to clearly appear in the trace. This is why a more powerful class of attacks, called *differential power analysis* (DPA), consist in analyzing a large amount of traces obtained during several executions of a cryptographic algorithm on different data (e.g. a different plaintext each time for an encryption scheme). Having multiple traces reduces the noise and running it on different inputs helps by showing the impact on consumption of the constant secret information (the key) under several circumstances. That being said, some attacks on a single (or low amount of) trace can be devastating, especially if a profiling phase capturing the behavior of the device is performed beforehand by the attacker (see template attacks [35]).

4.3.1 Correlation Power Analysis

We now describe a very powerful DPA attack that we performed on a block cipher called Kalyna. This attack, called *correlation power analysis* is a statistical approach aiming at finding a correlation between the expected variation of a device's power consumption for a given key hypothesis and the real variation of the consumption somewhere in the power traces, confirming the hypothesis.

Let us say that a device containing a (secret) key sk is running a cryptographic algorithm successively on a set of plaintexts $\mathcal{P} = \{p_1, p_2, ..., p_m\}$. During the i^{th} encryption, there should be an operation depending on both p_i and sk. Let us call it $f(p_i, sk)$. Given a set of power traces $\mathbf{T}$ measured while the device encrypts the plaintexts in $\mathcal{P}$, we aim to find a correlation between the variation of the power consumption at a certain point in the traces and the expected variation of the power consumption while performing $f(\cdot, \cdot)$, according to an appropriate leakage model. In order to do so, we precompute, for each element k_j of a set of key hypothesis $\mathcal{K}$, the expected power consumption $\mathbf{e_j} = \{L(f(p_1, k_j)), L(f(p_2, k_j)), ..., L(f(p_m, k_j))\}$ where $L(\cdot)$ is the leakage model

function. If we find j such that $\mathbf{e}_j$ correlates with a place in the traces, we can, with high confidence, say that $k_j = sk$. More precisely, we can split the process in three phases.

Measurement phase

We acquire power traces by measuring the consumption of a device running the cryptographic algorithm for a set of plaintexts $\mathcal{P} = \{p_1, p_2, ...p_m\}$. Each trace can be seen as a vector $\mathbf{t} \in \mathbb{R}^n$ with each element representing the instantaneous consumption at a given point in time and n depending on the measurement device. We end up with a matrix $\mathbf{T} \in \mathbb{R}^{m \times n}$ containing the data of all the measurements made.

Expected consumption estimation

First, we choose an operation $f(p, sk)$ depending on (a part of) the plaintext and on a part of the key. Then, for each possible key hypothesis k_j and each message $p_i \in \mathcal{P}$, we compute $L(f(p_i, k_j))$ with $L(\cdot)$ a function depending on the leakage model mapping the co-domain of $f(\cdot, \cdot)$ on a value $e \in \mathbb{R}$ representing the expected consumption of the device just after performing $f(p_i, k_j)$. Finally we store those values in a matrix $\mathbf{E} \in \mathbb{R}^{m \times |\mathcal{K}|}$ in which each column expresses the expected variation of the device's power consumption as a function of the plaintext, for a given key hypothesis.

Finding the key

In this last phase, we are going to find which key hypothesis corresponds to the real key used by the device. To do that, we look for a column $\mathbf{c_1}$ of $\mathbf{T}$ and a column $\mathbf{c_2}$ of $\mathbf{E}$ such that $\mathbf{c_1}$ and $\mathbf{c_2}$ correlates greatly. If we are able to find them, the key k_j that spanned $\mathbf{c_2}$ in $\mathbf{E}$ is the key sk used by the device with high probability. To evaluate the correlation, we usually simply use the sample correlation coefficient which associates to two S-size datasets $\mathcal{D} = \{d_1, d_2, ..., d_S\}$ and $\mathcal{D}' = \{d'_1, d'_2, ..., d'_S\}$ the value:

$$r = \frac{\sum_{i=1}^{S}(d_i - \overline{\mathcal{D}}) \cdot (d'_i - \overline{\mathcal{D}'})}{\sqrt{\sum_{i=1}^{S}(d_i - \overline{\mathcal{D}})^2 \cdot \sum_{i=1}^{S}(d'_i - \overline{\mathcal{D}'})^2}}$$

where $\overline{X}$ denotes the mean value of the dataset X.

4.4 Concrete attack: Breaking an Implementation of Kalyna ! (SPACE 2016)

We now move on to experimental results of our attack on our implementation of Kalyna. Since symmetric encryption is fairly off-topic of this thesis, we will not introduce all the notions required to understand the inner workings of the cipher. However, these are fairly standard (e.g. S-box) and accessible to anyone with a background in symmetric cryptography.

Kalyna

Kalyna is a SPN block cipher chosen as the new encryption standard of Ukraine during the Ukrainian National Public Cryptographic Competition [97, 98]. Kalyna is a Rijndael-like cipher based on five operations: substitutions i.e. S-boxes (SubBytes), rows shifting

(ShiftRows), column mixing (MixColumns), exclusive-or (XorRoundKey) and addition modulo 2^{64} (AddRoundKey).

Both the encryption and the key schedule use those five operations. The key schedule generates a couple of dependent round keys: round keys with even indices are generated by using the five transformations while round keys with odd indices are simple rotations of the previous (even) round keys. Those round keys are derived from an intermediate key K_t which is itself derived from the master key K. Kalyna supports block size and key size of 128, 256 and 512 bits with the key length equal or double of the block size. Kalyna can be referred as Kalyna b/k where b and k denote the block size and the key size (in bits).

4.4.1 Encryption Algorithm

In this section we will describe Kalyna 128/128 encryption algorithm with block size and key size of 128 bits which is the version we attacked. Kalyna 128/128 algorithm is summarized in Figure 4.1. The 128-bit inputs (round keys and ciphertext) of Kalyna 128/128 are represented with matrices in $GF(8)^{8\times 2}$ in an AES-like fashion. We simply call STATE the internal matrix containing the plaintext before encryption.

AddRoundKey

At the beginning and at the end of the encryption, the round key is added to STATE. The addition is a 2^{64} modular addition where each column (interpreted of the round key is added to each column of STATE.

SubBytes

Four S-boxes are used in Kalyna. Each byte $s_{i,j}$ of STATE is passed through the corresponding S-box: $s_{i,j} = SBox_{(i \mod 4)}(s_{i,j})$ with $0 \leq i \leq 7$ and $0 \leq j \leq 1$.

ShiftRows

This operation depends on the block size. For Kalyna 128/128 the four last bytes of each column of STATE are swapped together.

MixColumns

The bytes of each column of STATE are linearly combined by multiplying them with an 8×8 MDS matrix over $GF(2^8)$ which is obtained by rotating the vector $v = (0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04) \in GF(2^8)^8$. Unlike AES, the MixColumns operation takes place in every round of the encryption.

XorRoundKey

Finally, at the end of each round (except for the final round), the round key is bitwise XOR-ed with STATE.

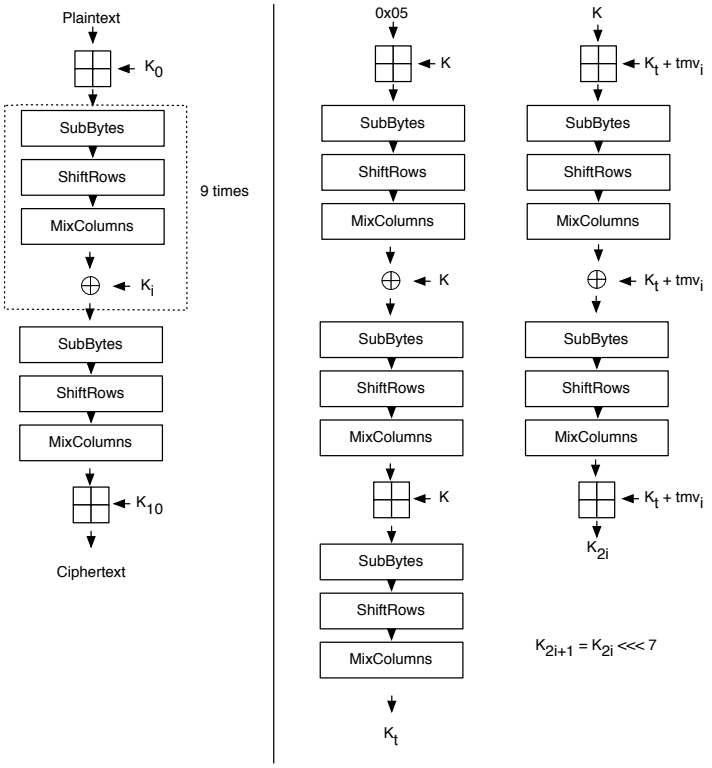

Figure 4.1: Kalyna 128/128 encryption scheme (left part) and Kalyna 128/128 key schedule (right part).

4.4.2 Key Scheduling

Kalyna uses a two steps key schedule. During the first step, K_t is computed from the master key K.

In the second step, K and K_t are used with another value tmv_i:

$$\begin{cases} tmv_0 = 0x01000100..0100 \\ tmv_{i+2} = tmv_i << 1 \end{cases}$$

where "$<< 1$" corresponds to a binary left shift of one position and .. is a padding of zeroes. Round keys with even indices are generated with round operations while round keys with odd indices are generated by *rotating* the previous (even) round key in the following way: $k_{2i+1} = k_{2i} <<< 7$.

Figure 4.1 summarizes Kalyna 128/128 key schedule. This key schedule does not allow better recovery of the master key from a round key than brute force [3].

4.4.3 Side-Channel Attacks on Kalyna

The goal of an attacker during a side-channel attack on a block cipher scheme is to get the master key that is used to generate all round keys during the encryption. For modern block cipher like DES and AES, an attacker that knows a set of plaintexts would focus the attack on the output of the S-box of the first round in order to recover the first round key. In case of ciphers such as AES, DES or Present [25] it is easy to reconstruct the master key from its round keys. For example, in AES-128 the first round key is the master key

and in Present-80 the first round key immediately gives us 64 most significant bits of the master key. In some cases it is necessary to consider more that the first round key, e.g., in AES-256 the second round key has also to be targeted in order to get the second half of the master key. In the case of DES, an attacker would have to brute force one byte of a key (because of the compression function *Permutation Choice*) in order to get the master key from the extracted round key.

Kalyna block cipher does not allow to easily run the key scheduling algorithm backwards and it is not using its master key directly in the encryption process as one of the round keys. Thus, getting the master key from round keys of Kalyna cipher is not as easy as for other commonly used ciphers such as mentioned above. This property makes Kalyna an interesting case-study for side-channel attacks.

The attacker has a choice between targeting K (the master key), K_t (the derived master key that is used to generate round keys) or to target directly the round keys K_i. All of these values might be targeted using a profiled attack in a usual way (as for any other block cipher) and it would be easier for an attacker to directly target K in order to extract the entire master key at once. However it implies that the attacker has to build a profile on the basis of a similar device. Here, we try to avoid this constraint.

Considering non-profiled attacks and the particular case of Kalyna, the easiest target seems to be the round keys because they are used with different inputs (different plaintexts).

In the following, we are considering a scenario where an attacker is not able to mount a profiled attack on the device and we are focusing on classical non-profiled CPA. Our CPA on Kalyna targets all round keys K_i.

CPA on Kalyna

Our approach was to apply a classical CPA attack round by round. We use the Hamming weight model as leakage model and attack the output of the S-boxes. Once a round key is found the solution is used for attacking the next round: computing Kalyna cipher until the next round with all the guessed round keys. A great property of Kalyna for our attack is that we can use the round keys dependency to improve and verify our guesses: if k_{2*i+1} is not equivalent to k_{2*i} with the rotation then k_{2*i} must have been incorrectly guessed. For the first round key we have also to take the carry bit into account: due to the fact that the cipher starts with a modular addition, the CPA attack not only have to consider the message p_i and the key hypothesis k_j but also the carry bit cb when computing the leakage $L(f(p_i, k_j, cb))$. Thus the attack on one byte of the round key of Kalyna depends on the attack on the previous byte (except the first byte of each column). This property forces the attacker to perform more computations during the attack. When all the keys are retrieved, the attack succeeds the same way as it had retrieved the master key since having those keys is sufficient to decrypt.

4.4.4 Experiments

Data acquisition setup

For our experiments we have implemented Kalyna on a 8-bit microcontroller ATMega 328. We used a fixed key and random plaintexts during the execution of the algorithm. We acquired 1000 power traces using Infiniium MSO9254A oscilloscope that was set up to acquire 200 MSamples/s. Each power trace is the average of 64 single acquisitions, the

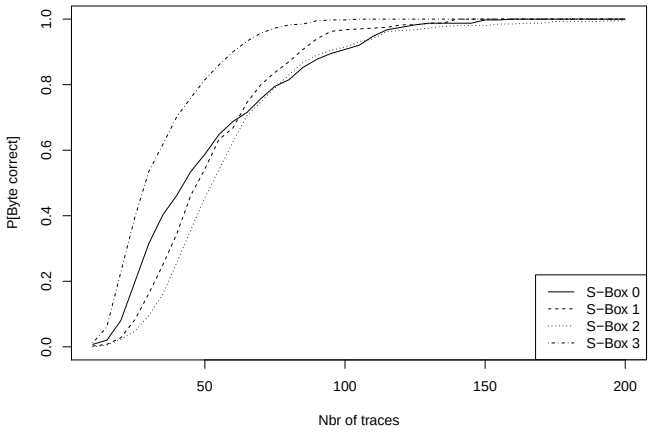

Figure 4.2: Average success rate of the four S-boxes when attacking K_0.

averaging was done by the oscilloscope in order to reduce noise. A small $10\ \Omega$ resistor was placed between the group of our 5 Volt power supply and the ground pin of the microcontroller in order to do the acquisitions.

Our attack

Based on a set of traces, we begin the attack with the bytes of K_0. The first roundkey is a particular key to attack since we have to take the carry into account[1] (except for the first byte of each column). Once we have a complete guess for K_0 and K_1 we must ensure that they match (i.e. K_1 is a rotation of K_0). If K_0 and K_1 do not match, we must consider adding more traces to the set and start again. Otherwise we can focus on roundkeys 2 to 9. Each time a pair of roundkeys is found we verify whether they match or not (starting the attack again in case they do not). For the final roundkey K_{10}, we execute the cipher until the final AddRoundKey operation and retrieve it based on the computation and the ciphertext. Algorithm 20 summarizes our attack.

CPA results

Figure 4.2 shows the average[2] probability of success of a practical attack on the four S-boxes for the first roundkey (K_0). We performed our attack with a growing set of traces in order to experimentally estimate the success probability in function on the number or traces used. We did 100 attacks for each number of used traces. Each of those attacks were executed with randomly chosen traces (out of a set of 1000 traces).

Figure 4.3 shows the average (from 100 experiments) success rate of the entire attack: retrieving all the 10 keys, K_0 to K_9 (the last round key can be retrieved from the ciphertext and the execution of the algorithm until the last AddRoundKey operation). With 250 traces the success rate of the attack is 96%.

[1] An error propragation of the carry will be noticed at the end of K_1 recovery and will result in a mismatching of K_0 and K_1. This will imply restarting the recovery of K_0 with more traces.

[2] Since is each S-box is used four times for each round we averaged the results of each S-box for the same round.

Algorithm 20 Pseudo-code of CPA attack on Kalyna 128/128

Require: Set of traces, ciphertext
Ensure: Roundkeys K_0 to K_{10}

```
                                                          ▷ Attacking K_0 and K_1
 1:
 2: for i ← 0, 1 do
 3:     CPA on byte 0 of column i of K_0
 4:     for j ← 1, 7 do
 5:         CPA on byte j of column i of K_0 using guessed byte (j − 1)
 6:     end for
 7:     for i ← 0, 1 do
 8:         for j ← 0, 7 do
 9:             CPA on byte j of column i of K_1
10:         end for
11:     end for
12: end for
13: if K_1 is not a rotation of K_0 then
14:     add more traces to the set of traces
15:     restart attack of K_0 and K_1
16: end if
17:                                                       ▷ Attacking K_2 to K_9
18:
19: for k ← 1, 4 do
20:     for i ← 0, 1 do
21:         for j ← 0, 7 do
22:             CPA on byte j of column i of K_{2*k}
23:         end for
24:     end for
25:     for i ← 0 to 1 do
26:         for j ← 0 to 7 do
27:             CPA on byte j of column i of K_{2*k+1}
28:         end for
29:     end for
30:     if K_{2*k+1} is not a rotation of K_{2*k} then
31:         add more traces to the set of traces
32:         restart attack of K_{2*k} and K_{2*k+1}
33:     end if
34: end for
35:                                                       ▷ Attacking K_10
36:
37: pre_cipher ← execute encryption until final AddRoundKey using K_0 to K_9
38: from ciphertext and pre_cipher compute K_10
```

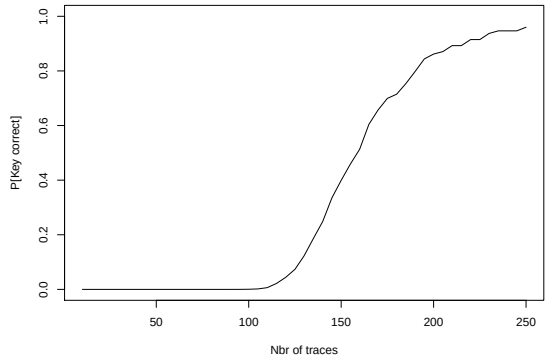

Figure 4.3: Average success rate of the entire attack as a function of the number of traces.

4.5 Masking

Our real-life attack of last section showed that side-channel attacks are doable in practice. We will now discuss a potential countermeasure against it called masking. The goal of a masking scheme is to randomize the values handled by the device during the execution of a cryptographic algorithm to blind the adversary. For example, in our attack on Kalyna, the successful recovery of a key byte was severely linked to the reuse of the same key for each encryption. Indeed, the correlation was computed between real and estimated consumption measurements *for a given key (byte) hypothesis*. If this key byte is randomized at each execution, the attack fails. Of course refreshing the key each time is unsatisfactory, so we have to find a way to randomize values while keeping the final result mathematically equivalent to the original scheme. In a masked scheme, sensitives values v are split in multiples values v_i called shares. The idea is that if the adversary successfully recovers one of the v_i, it does not help to recover v. Actually, even recovering all the v_i but one should not help. This method is an example of something known in the cryptographic literature under the name *secret sharing*. At each execution, the v_i will be randomized such that they successively take different values while their combination stays v. Simply splitting each key byte in two shares would be devastating for our attack on Kalyna since it targets a specific instant in the trace. However, there exists more sophisticated attacks that target multiple points at the same time called high-order attacks. This is why, in practice, values might be split in more than two shares. When sensitive values are split in $N + 1$ shares, we call it a masking of order N.

Arithmetic and Boolean masking

Let us introduce two types of additive combination in the following definition.

Definition 38 (Arithmetic and Boolean masking)**.** *A sensitive value x is shared with mod q arithmetic masking if it is split into $N + 1$ shares $(x_i)_{0 \leq i \leq N}$ such that*

$$x = x_0 + \cdots + x_N \pmod{q}. \qquad \text{(Arithmetic masking mod } q\text{)}$$

It is shared with Boolean masking if it is split into $N + 1$ shares $(x_i)_{0 \leq i \leq N}$ such that

$$x = x_0 \oplus \cdots \oplus x_N. \qquad \text{(Boolean masking)}$$

More generally, if our value x can be encoded as a group element for any group $(G, \circ)$, a masking of order N is a tuple $(x_i)_{0\leq i\leq N} = (x_0, x_1, \ldots, x_n)$ such that $x = x_0 \circ x_1 \cdots \circ x_n$. In our case, we use the arithmetic masking in $\mathbb{Z}_q$ because the main mathematical objects of the scheme are polynomials in $\mathbb{Z}_q[X]/\langle X^n+1\rangle$ and the Boolean masking which represent values as a direct sum of $GF(2)$ because it is the representation used by computers. We call $(x_i)_{0\leq i\leq N}$ the masked value of x.

Gadgets

A masking scheme is an implementation of a cryptographic scheme that performs its computations on masked sensitive values. In the case of a signature scheme, it takes as input a masked secret key and an unmasked message (because it is public and known to the adversary) and outputs an unmasked signature. The important point is that all sensitive values used during the execution of the algorithm must be masked. Identifying which intermediate values need to be kept in masked form and which can be safely unmasked might be challenging in some cases since designers do not study the security of their scheme "from the inside". It is thus required to design subroutines implementing the functionality of the scheme without unmasking its inputs. Such subroutines are called gadgets.

Definition 39. *A (u, v)-gadget is a probabilistic algorithm that takes as inputs u shared values, and returns randomized v-tuples of shared values.*

The output is said to be randomized because even if the value resulting from the combination of the shares is the same at each execution [3], the value of each individual share will change. In practice, we will sometimes take the liberty to call "gadget" any algorithm, even not randomized, that performs operations on masked values.

4.6 Lattice-Based Signatures

Our second work presented in this chapter is a masking scheme for the NIST candidate qTESLA. We decided to mask this signature because a similar work on its sibling DILITHIUM had already been published and thus masking qTESLA would enable fairer comparisons in the framework of the standardization project. This signature scheme is the result of several iterations over a scheme of Lyubashevsky [80, 81] constructing a lattice version of Schnorr signature. Thus, we will now describe how this type of signature works and quickly explain the different variants that appeared in the literature.

4.6.1 GLP

For the sake of brevity, we will not describe in details the earliest versions of these signatures but directly jump to a version illustrating the concept. This version, called the GLP [63] signature scheme, is now mainly considered deprecated but is actually the first one that has been masked. In the following, we use the notation $\mathcal{R}_{q,[k]}$ to denote polynomials in $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ with coefficients in $[-k, k]$.

[3] This is not always the case because the functionality implemented by the gadget might also be randomized, e.g. the generation of a masked key.

Public parameter: a
Secret key: $s, e \xleftarrow{r} \mathcal{R}_{q,[1]}$
Public key: $t \leftarrow a \cdot s + e$
Random oracle $H : * \rightarrow \{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = 32\}$

Sign(s, e, m):

1: **do**
2: $y_1, y_2 \xleftarrow{r} \mathcal{R}_{q,[B]}$
3: $c \leftarrow H(a \cdot y_1 + y_2, m)$
4: $z_1 \leftarrow s \cdot c + y_1$, $z_2 \leftarrow e \cdot c + y_2$
5: **while**$\big(z_1 \notin \mathcal{R}_{q,[B-32]}$ **or** $z_2 \notin \mathcal{R}_{q,[B-32]}\big)$
6: **return** z_1, z_2, c

Verify(z_1, z_2, c, t, m):

1: $v \leftarrow a \cdot z_1 + z_2 - t \cdot c$
2: **return** 1 **if** $c = H(v, m)$ **and both** z_i **are small else** 0

Figure 4.4: GLP signature scheme.

Decisional Compact Knapsack

To argue security and offer an efficient implementation, the authors used a small parameters version of decisional RLWE called the decisional Compact Knapsack problem (DCK). In that version, the secret and error distributions are $\mathcal{U}(\{-1, 0, 1\})$ which means that the adversary receives tuples of the form $(a, a \cdot s + e)$ with $a \xleftarrow{r} \mathcal{R}_q$ and $(s, e) \xleftarrow{r} (\mathcal{R}_{q,[1]} \times \mathcal{R}_{q,[1]})$, and must distinguish them from samples from $\mathcal{U}(\mathcal{R}_q \times \mathcal{R}_q)$. One can also naturally define the corresponding search problem. This is simply a variant of RLWE but we keep the nomenclature of the literature for consistency.

The scheme

The scheme is described in Figure 4.4. The public key is a simple DCK sample and the private key consists in the corresponding secret and error polynomials. We emphasis that unlike the RLWE key exchanges exposed earlier, the error vector is part of the private key and is required to sign. The Fiat-Shamir structure of the signature is clear and the underlying identification scheme acts as a proof of knowledge of values s and e such that $t = as + e$. Similarities with the Schnorr signature described in Figure 2.5 are obvious. The commitment is an instance of the underlying hard problem that is recovered by the verifier, the challenge is the output of the random oracle on the instance and the message to sign and the signature is of the form $z = sc + y$. As such, it could be simply seen as Schnorr over DCK instead of the discrete logarithm. However, we observe two crucial differences:

- the signature is composed of two values z_1, z_2 and,
- the signature process has to be restarted if each signature component does not fall in a certain range.

The first observation is easily explained by the fact, unlike for the discrete logarithm, the secret is composed of two values s and e. Hence, values depending on both s and e have to be transmitted to the verifier for him to recover exactly the commitment $a \cdot y_1 + y_2$. The second one is the implementation of a mechanism called rejection sampling. The value B is a parameter of the scheme that gives a trade-off between security and effiency. If B is small, the acceptance probability will be low and the signature procedure will have

to restart often. If B is large, an adversary can attempt to forge a signature by picking a random v, computing $c = H(v, m)$ and trying to find z_1, z_2 such that $v = a \cdot z_1 + z_2 - t \cdot c$. Indeed, the hardness of this problem actually depends on the shortness of (z_1, z_2).

Rejection sampling

Obviously, we do not want an adversary to learn anything about the private key from the signature. In the case of Schnorr signature, we made the following reasoning in Section 2.2.9: $z = sc + y$ does not help to recover s because y is uniformly random and unknown to the adversary, hence offering perfect secrecy on s. Also, the value y is kept unknown because even though the adversary can compute the commitment g^y, the discrete logarithm is assumed to be hard. Similarly here, to have any security, it should be computationally hard to compute the secret key (s or e) from any public value. Unfortunately, the reasoning above fails. While it is also true that from the commitment $ay_1 + y_2$, it is computationally hard to find y_1 and/or y_2 under the DCK assumption, the signature components $z_1 = sc + y_1$ and $z_2 = ec + y_2$ do not offer perfect secrecy on s and e because the computations are taking place in the ring $\mathcal{R}_q$ while the y_i are sampled from a subset of $\mathcal{R}_q$ and thus, does not perfectly hide the products. Indeed, we showed in the introduction that the Generalized One-time pad offers perfect secrecy only if the "key" is sampled from a uniform distribution over the group (which would be the additive group of $\mathcal{R}_q$ in our case). Let us simply focus on an expression of the form $z = sc + y$ since z_1 and z_2 are equivalent or this issue. The problem is that the distribution of $sc + y$ depends on the value of the product sc. For example, writing $(p)_j$ the j-th coefficient of a polynomial p, if $(z)_j = B + 32$, the adversary directly learns that $(y)_j = B$ and $(sc)_j = 32$, which are both sensitive values. This problem would not appear with a uniform $(y)_j$ because for any $(z)_j$ there would be as many possible $(y)_j$ as there are possible $(sc)_j$. The solution proposed in the signature is to output z if and only if its coefficients are values that are possible *for any* product sc. In our case, if all the coefficients of z are in the interval $[-B + max(\|sc\|_\infty)), B - max(\|sc\|)_\infty)]$, they all appear with equal probability, independently of the value sc. From the point of view of the adversary, the coefficients of z will be distributed according to a uniform distribution over $[-B + max(\|sc\|_\infty, B - max(\|sc\|_\infty]$. Figure 4.5 graphically illustrates this phenomenon. In the case of GLP, $max(\|sc\|_\infty) = 32$ since $\|c\| = 32$ and s has coefficients in $\{-1, 0, 1\}$. The reader might wonder why not simply pick uniform y_1, y_2, this is because the scheme bases its security on the shortness of z_1, z_2. Otherwise, the adversary could trivially pick random v, z_1, set $z_2 \leftarrow v - az_1 + tc$, $c \leftarrow H(v, m)$ and compute the forgery z_1, z_2, c. The rejection sampling procedure we just described is tailored for uniform distributions. The general case is covered by a nice lemma of [81]:

Lemma 1. *Let V be an arbitrary set, and $h : V \to \mathbb{R}$ and $f : \mathbb{Z}^m \to \mathbb{R}$ be probability distributions. If $g_v : \mathbb{Z}^m \to \mathbb{R}$ is a family of probability distributions indexed by all $v \in V$ with the property that*

$$\exists M \in \mathbb{R} \text{ such that } \forall v, Pr[Mg_v(z) \geq f(z); z \xleftarrow{r} f] \geq 1 - \epsilon$$

then the distribution of the output of the following algorithm $\mathcal{A}$:

1. $v \xleftarrow{r} h$
2. $z \xleftarrow{r} g_v$
3. *output* (z, v) *with probability min* $\left(\frac{f(z)}{Mg_v(z)}, 1\right)$

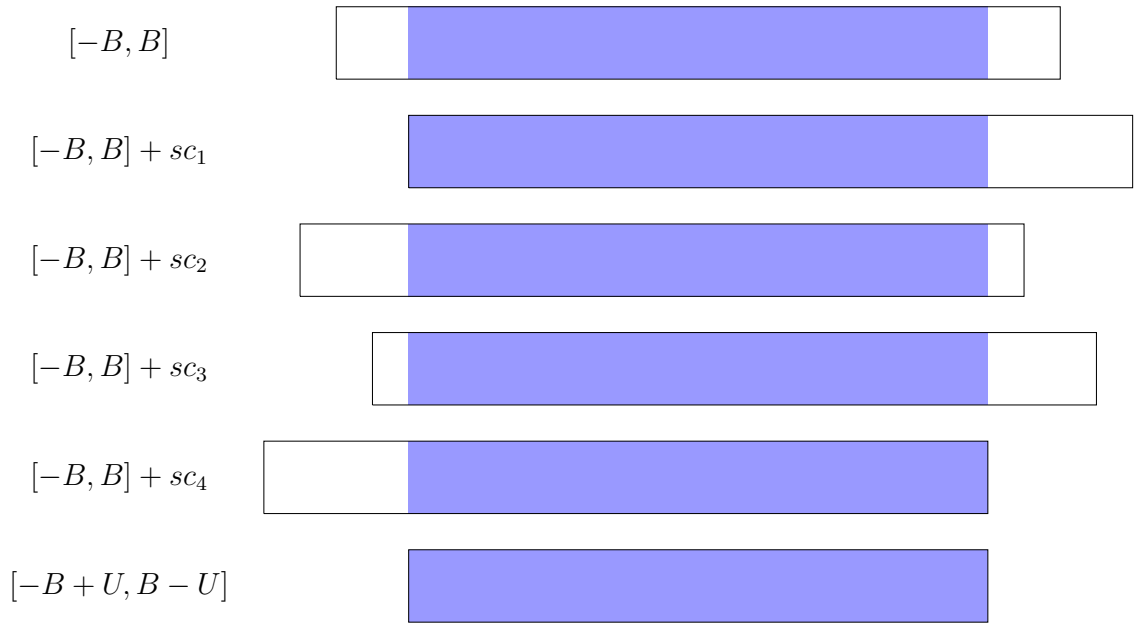

Figure 4.5: Rejection sampling. The distribution of the z during the signing procedure is the uniform distribution over $[-B, B]$ shifted by some sc. The signature is output only if z falls into the colored interval, otherwise the procedure is restarted. Thus, from the point of view of the adversary, the signature are uniform values in $[-B + U, B - U]$. The constant U is chosen as the maximum of the product sc in absolute value.

is within statistical distance[4] ϵ/M *of the distribution of the following algorithm* $\mathcal{F}$*:*

1. $v \xleftarrow{r} h$
2. $z \xleftarrow{r} f$
3. *output* (z, v) *with probability* $1/M$

Moreover, the probability that $\mathcal{A}$ *outputs something is at least* $(1 - \epsilon)/M$*.*

Basically, this lemma extends the technique to more general probability distributions. It introduces the notion of a scaling factor M that has to be chosen such that $f(z) \leq Mg_v(z)$, for all v. For distributions with infinite support, such an M might not exist. This is why the lemma also introduces an ϵ variable that let the output distribution be statistically close to but not exactly the same as the targeted distribution. In practice, ϵ should be chosen such that it is cryptographically negligible. In the case of GLP (for one coefficient of the polynomials), g_v is the family of uniform distributions over $[-B, B]$ shifted by the different possible sc and $M = \frac{2B+1}{2(B-32)+1)}$. The probability of acceptance for the whole z_1 and z_2 is thus $(\frac{1}{M})^{2n} = (1 - \frac{64}{2B+1})^{2n}$.

Signature compression

The authors of GLP observed that a compression algorithm could be applied to their scheme in order to reduce signature size. In particular, using their notation, they propose to rewrite a polynomial p in a low part

$$p^{(0)} = p \mod (2(B - 32) + 1)$$

[4] simply defined as $\Delta(\mathcal{A}, \mathcal{B}) = \frac{1}{2} \sum_{x \in X} |\mathcal{A}(x) - \mathcal{B}(x)|$ for two distributions $\mathcal{A}$ and $\mathcal{B}$ over X

Public parameter: $\mathbf{A} \xleftarrow{r} \mathbb{Z}_q^{m \times n}$
Secret key: $\mathbf{S} \xleftarrow{r} \chi_\sigma^{n \times k}$
Public key: $\mathbf{T} \leftarrow \mathbf{AS} + \mathbf{E} \mod q$ with $\mathbf{E} \xleftarrow{r} \chi_\sigma^{m \times k}$ such that $|\mathbf{E}_{i,j}| > 7\sigma$ for all (i, j)

Sign$(m, \mathbf{S}, \mathbf{T})$:

$\mathbf{y} \xleftarrow{r} \mathcal{R}_{q,[B]}$
$\mathbf{v} \leftarrow \mathbf{Ay} \mod q$
$c \leftarrow H(\lfloor \mathbf{v} \rceil_d, m)$
$\mathbf{c} \leftarrow F(c)$
$\mathbf{z} \leftarrow \mathbf{y} + \mathbf{Sc}$
$\mathbf{w} \leftarrow \mathbf{Az} - \mathbf{Tc} \mod q$
if $|[\mathbf{w}_i]_{2^d}| > (2^{d-1} - 7\omega\sigma)$
    restart
if $\mathbf{z} \notin \mathcal{R}_{q,[B-U]}$
    restart
return $\mathbf{z}, c$

Verify$(\mathbf{T}, \mathbf{A}, \mathbf{z}, c, m)$:

$\mathbf{c} \leftarrow F(c)$
$\mathbf{w} \leftarrow \mathbf{Az} - \mathbf{Tc}$
$c' \leftarrow H(\lfloor \mathbf{w} \rceil_d, m)$
return $(c' = c)$ **and** $(\mathbf{z} \in \mathcal{R}_{q,[B-U]})$

Figure 4.6: Bai-Galbraith signature

and a high part

$$p^{(1)} = (p - p^{(0)})/(2(B - 32) + 1).$$

And give in the appendix of the paper an algorithm $z' \leftarrow \mathsf{Compress}(y, z)$ such that $(y + z)^{(1)} = (y + z')^{(1)}$ and z' admits a short representation. Then they propose that the signing procedure uses $(ay_1 + y_2)^{(1)}$ as commitment and outputs $(z_1, z_2' = \mathsf{Compress}(az_1 - tc, z_2))$ instead of (z_1, z_2). This technique greatly reduces the signature size without affecting too much its efficiency. To this day, the original parameter sets still offer decent performances. However, the advances in cryptanalysis, the superior design of the schemes we present next and the concerns about the DCK ([117]) makes it a deprecated algorithm.

4.6.2 A Scheme from Bai and Galbraith

The next important scheme we introduce is due to Bai and Galbraith [15] and can be found in Figure 4.6. We will call it the BG signature in the following. It is defined over LWE with public keys of the form $\mathbf{T} = \mathbf{AS} + \mathbf{E}$ in which $\mathbf{S}$ and $\mathbf{E}$ are also matrices (but with small entries). Since both the secret and the error are matrices, the public key should be seen as multiple parallel instances of LWE. Starting from the compressed GLP signature, the core of their design is to go one step further by completely removing the z_2 part from the signature. The argument is that it is important to prove the knowledge of an $\mathbf{S}$ such that $\mathbf{T} = \mathbf{AS} + \mathbf{E}$ for a short $\mathbf{E}$ but the exact value of $\mathbf{E}$ is irrelevant. The scheme is parameterized by q, B, n in the same way as GLP but also adds some extra parameters:

- m and k control the number of sample and the number of parallel instances of LWE;
- σ is the standard deviation of the discrete Gaussian distributions used to sample the secret and error matrices;
- d is the number of bits removed while rounding (see below);

- ω is the weight of the challenge vector (it is the same as the 32 constant of GLP);
- $U = 14\sigma\sqrt{\omega}$ is a bound on the value of $\mathbf{Sc}$.

The algorithm is eventually rather similar to the compressed version of GLP in its functioning. However, the high part extraction and compression are replaced by a rounding function and a "well rounded" verification.

Rounding function and well rounded verification

For an integer v, let us write $[v]_{2^d}$ the representative in $(-2^{d-1}, 2^{d-1}]$ of v modulo 2^d. The rounding of an integer x is defined as

$$\lfloor x \rceil_d = (x - [x]_{2^d})/2^d.$$

This high order bits extraction is used to compute the commitment of the signature which is $\lfloor \mathbf{Ay} \mod q \rceil_d$. During signature verification, an approximate value $\mathbf{Az} - \mathbf{Tc} \approx \mathbf{Ay}$ is computed and will correspond to the commitment after rounding. To avoid having cases in which $\lfloor \mathbf{Ay} \rceil_d \neq \lfloor \mathbf{Az} - \mathbf{Tc} \rceil_d$, an extra well rounded check is performed at line 7 during signing: if, for any coefficient w of $\mathbf{Az} - \mathbf{Tc}$, $|[w]_{2^d}|$ is greater than a fixed threshold, the signing procedure is restarted.

Posterity

The BG signature is the basis of most efficient Fiat-Shamir lattice-based signatures proposed to this day[5]. Optimization, new parameters and an efficient software implementation where presented in [43]. Later, the TESLA family appeared with signatures using standard lattices [9] and, for more efficiency, ideal lattices [2,16]. Following the discovery of a flaw in the security proof, the original TESLA scheme was revisited in [10]. The two most up-to-date schemes are the ones currently candidates in the NIST standardization project:

- qTESLA, the current iteration of the TESLA family, for which we will soon describe a masking scheme as main contribution of this chapter;
- DILITHIUM, which is a MLWE derivative of BG/GLP that has been designed by a team close to the one of KYBER[6].

4.6.3 qTESLA

We now describe qTESLA, a (family of) lattice-based signature based on the RLWE problem and round 2 candidate for the NIST's post-quantum competition. Its design is basically the same as the one of Bai-Galbraith signature, but since the masking of qTESLA is the main contribution of this chapter, we explain it in greater details here. When we were working on the masking scheme, the qTESLA submission was proposing two different type of parameters sets: heuristic and provably secure. The provably secure parameter sets offered a tight security proof based on the hardness of decisional

[5]One notable exception might be the BLISS signature scheme [49] which is closer to the original design of Lyubashevsky but is somewhat cumbersome to implement securely.

[6]Both of them are actually part of "Cryptographic suite" called CRYSTALS. `https://pq-crystals.org/index.shtml`

RLWE in the *quantum* random oracle model (QROM), that is to say a model in which the adversary can query the random oracle on quantum states. On the other hand, the heuristic parameter sets aimed to have shorter and faster signatures that base their security on post-quantum assumptions in the classical random oracle model as it is done in GLP/BG/DILITHIUM. Since the usefulness of the quantum random oracle model is still not clear, is not used in the direct rival DILITHIUM (however it is possible to instanciate it in the QROM, see [72]) and hurts performances, we decided to focus on the heuristic version. While our work was in the peer review phase of CARDIS 2019, the QTESLA team removed the heuristic parameters from its submission to focus on the provably secure instanciation. Later, our work was accepted to the conference and thus we present experimental results on unsupported parameter sets. However, we emphasis that the parameters were not broken, only removed from the submission and that the masking scheme does not depend on the set used, all the contributions on the masking side are still very relevant and applicable to the provably secure version. Nonetheless, even though the scaling at high masking orders is expected to be the same, the lightest provably secure parameter set is (according and old version of the specification document) around 6 times slower, outputs signatures 2 times larger and has a public key 10 times larger than the lightest heuristic set. It would thus be unlikely to chose this version over DILITHIUM in a small device prone to side-channel attacks.

QTESLA Parameters

The parameters used in the signature are really similar to the ones of GLP and BG, but we recall them for completeness:

- n: Dimension of the ring
- q: Modulus
- σ: Standard deviation of the discrete gaussian
- h: Number of nonzero entries of the polynomial c
- E and S: Rejection parameters
- B: Bounds for the coefficients of the hiding polynomial y
- d: Number of bits dropped in rounding (used in the computation of $\lfloor \cdot \rceil_M$)
- δ: overall acceptance probability

The two parameter sets we used in our experiments are listed in Table 4.1. We use the names QTESLA-I and QTESLA-III as they used to appear in the original submission.

Scheme

Hereunder will be explicitly described the main algorithms, namely key generation, sign and verify. Beforehand, let us briefly recall the functionality of each of the subroutines and functions for completeness. We redirect the interested reader to [7] or the NIST submission for a detailed description.

Parameters	qTESLA-I	qTESLA-III
n	512	1024
q	4 205 569 $\approx 2^{22}$	8 404 993 $\approx 2^{23}$
σ	22.93	10.2
h	30	48
E	1586	1147
S	1586	1233
B	$2^{20} - 1$	$2^{21} - 1$
d	21	22
δ	0.16	0.24

Table 4.1: Parameters for qTESLA-I and qTESLA-III

- PRF: Pseudorandom function, used to expand a seed into arbitrary size randomness.
- GenA: Generates a uniformly random polynomial $a \in \mathcal{R}_q$.
- GaussSampler: Sample a polynomial according to a gaussian distribution, parameters of the distribution are fixed in the sampler.
- CheckS: Verify that the secret polynomial s does not have too large coefficients.
- CheckE: Verify that the secret polynomial e does not have too large coefficients.
- ySampler: Sample a uniformly random polynomial $y \in \mathcal{R}_{q,[B]}$.
- H: Collision resistant hash function.
- Enc: Encode a bitstring into a sparse polynomial $c \in \mathcal{R}_{q,[1]}$ with $||c||_1 = h$
- $[\cdot]_L : \mathbb{Z} \to \mathbb{Z}, w \mapsto w \bmod^{\pm} 2^d$
- $[\cdot]_M : \mathbb{Z} \to \mathbb{Z}, w \mapsto (w \bmod^{\pm} q - [w]_L)/2^d$

Key generation (Algorithm 21)

The key generation will output a RLWE sample together with some seeds used to generate public parameters and to add a deterministic component to the signing procedure. The algorithm starts by expanding some randomness into a collection of seeds and generates the public polynomial a before moving on to the two secret values s and e. Those two values are sampled from a Gaussian distribution and have to pass some checks to ensure that the products $s \cdot c$ and $e \cdot c$ do not have too large coefficients. After that, the main component t of the public key is computed as $t = a \cdot s + e$. The output consists of the secret key $sk = (s, e, \text{seed}_a, \text{seed}_y)$ and the public key $pk = (seed_a, t)$.

Sign (Algorithm 22)

The signing procedure takes as input a message m and the secret key sk and outputs a signature $\Sigma = (z, c)$. First, in order to generate the randomness needed in the algorithm, a seed is derived from a fresh random value r, seed_y and m. Next, a polynomial

$y \in \mathcal{R}_{q,[B]}$ is sampled to compute the value $v = a \cdot y \bmod^{\pm} q$. The algorithm will now hash the rounded version of v together with the message and encode the result in a sparse polynomial c with only h entries in $\{-1, 1\}$. The candidate signature is computed as $z = y + s \cdot c$. Before outputting the result, two additional checks must be performed: we must ensure that z is in $\mathcal{R}_{q,[B-S]}$ and that $w = v - e \cdot c \bmod^{\pm} q$ is well rounded, meaning that $||[w]_L||_\infty < 2^{d-1} - E$ and $||w||_\infty < \lfloor q/2 \rfloor - E$ should hold. When one of the check fails, the signing procedure is restarted by sampling a new y. When eventually both checks pass, the signature $\Sigma = (z, c)$ is output.

Verify (Algorithm 23)

Signature verification is pretty lightweight and straightforward for this type of signature. Taking as input the message m, signature $\Sigma = (z, c)$ and public key $pk = (seed_a, t)$, it works as follow: First, it generates the public parameter a, then computes $w = a \cdot z - t \cdot c$ and accepts the signature if the two following conditions hold:

1. $z \in \mathcal{R}_{q,[B-S]}$
2. $c \neq \mathsf{Enc}(\mathsf{H}([w]_M, m))$

Algorithm 21 qTESLA key generation

Result: Secret key $sk = (s, e, \text{seed}_a, \text{seed}_y)$, public key $pk = (seed_a, t)$

counter $\leftarrow 1$
pre-seed $\xleftarrow{r} \{0,1\}^\kappa$
$\text{seed}_s, \text{seed}_e, \text{seed}_a, \text{seed}_y \leftarrow \mathsf{PRF}(\text{pre-seed})$
$a \leftarrow \mathsf{GenA}(\text{seed}_a)$
do
    $s \leftarrow \mathsf{GaussSampler}(\text{seed}_s, \text{counter})$
    counter $\leftarrow$ counter + 1
while ($\mathsf{CheckS} \neq 0$)
do
    $e \leftarrow \mathsf{GaussSampler}(\text{seed}_s, \text{counter})$
    counter $\leftarrow$ counter + 1
while ($\mathsf{CheckE} \neq 0$)
$t \leftarrow a \cdot s + e \bmod q$
$sk \leftarrow (s, e, \text{seed}_a, \text{seed}_y)$
$pk \leftarrow (seed_a, t)$
return sk, pk

4.7 Masking qTESLA at any Order (CARDIS 2019)

Due to their newness, lattice-based signatures have not been extensively studied from a side-channel point of view. Even though quite a few attacks appeared in the literature [23, 33, 34, 51, 52, 103], less has been done on the side of countermeasures. On the very specific domain of masking Fiat-Shamir lattice-based signatures, only two papers are

Algorithm 22 qTESLA sign

Data: Secret key $sk = (s, e, \text{seed}_a, \text{seed}_y)$, message m
Result: Signature $\Sigma = (z, c)$

counter $\leftarrow 1$
$r \xleftarrow{r} \{0,1\}^{\kappa}$
rand $\leftarrow \mathsf{PRF}(\text{seed}_y, r, H(m))$
$y \leftarrow \mathsf{ySampler}(\text{rand, counter})$
$a \leftarrow \mathsf{GenA}(\text{seed}_a)$
$v \leftarrow a \cdot y \bmod^{\pm} q$
$c \leftarrow \mathsf{Enc}(\mathsf{H}([v]_M, m))$
$z \leftarrow y + s \cdot c$
if $z \notin \mathcal{R}_{q,[B-S]}$ **then**
    counter $\leftarrow$ counter + 1
    goto 4
end if
$w \leftarrow v - e \cdot c \bmod^{\pm} q$
if $||[w]_L||_\infty \geq 2^{d-1} - E$ **or** $||w||_\infty \geq \lfloor q/2 \rfloor - E$ **then**
    counter $\leftarrow$ counter + 1
    goto 4
end if
return (z, c)

Algorithm 23 qTESLA verify

Data: message m, signature $\Sigma = (z, c)$ and public key $pk = (seed_a, t)$
Result: *accept* or *fail*

$a \leftarrow \mathsf{GenA}(\text{seed}_a)$
$w \leftarrow a \cdot z - t \cdot c \bmod^{\pm} q$
if $z \notin \mathcal{R}_{q,[B-S]}$ **or** $c \neq \mathsf{Enc}(\mathsf{H}([w]_M, m))$ **then**
    return *fail*
end if
return *accept*

prior to our work. The first one [19] presents a masking of GLP that lays the foundations for this line of research. The second one [92] discusses the masking of the more recent DILITHIUM scheme and provides some practicals experiments. It was thus natural for us to take care of the other NIST candidate: QTESLA.

Our code is currently available at

`https://github.com/fragerar/Masked_qTESLA`.

4.7.1 Masking-Friendly Design

In the process of masking QTESLA, we decided to make slight modifications in the signing procedure in order to facilitate masking. The idea is that some design elements providing small efficiency gains may be really hard to carry on to the masked version and actually do even more harm than good. Our two main modifications are the modulus which is chosen as the closest power of two of the original parameter set and the removal of the PRF to generate the polynomial y.

Power of two modulus.

Modular arithmetic is one of the core component of plenty of cryptographic schemes. While, in general, it is reasonably fast for any modulus (but not necessarily straightforward to do in constant time), modular arithmetic in masked form is very inefficient and one of the bottleneck in term of running time. In [19], a gadget SecAddModp is defined to add two integers in boolean masked form modulo p. The idea is to naively perform the addition over the integers and to subtract p if the value is larger than p. While this works completely fine, the computational overhead is large in practice and avoiding those reductions would drastically enhance execution time. The ideal case is to work over $\mathbb{Z}_{2^n}$. In this case, almost no reductions are needed throughout the execution of the algorithm since overflowing operations naturally reduce modulo a (larger) power of two and, when explicitely needed, can be simply performed by applying a mask on boolean shares. The reason why working with a power of two modulus is not the standard way to instanciate lattice-based cryptography is that it removes the possibility to use the number theoretic transform (NTT) to perform efficient polynomial multiplication in $\mathcal{O}(n \log n)$. Instead, multiplication of polynomial has now to be computed using the Karatsuba/Toom-cook algorithm which is slower for parameters used in state-of-the-art algorithms. Nevertheless, in our case, not having to use the heavy SecAddModp gadget largely overshadows the penalty of switching from NTT to Karatsuba. Since modulus for both parameter sets were already close to a power of two, we rounded up to the closest one, i.e. 2^{22} for QTESLA-I and 2^{23} for QTESLA-III. This modification does not change the security of the scheme. Indeed, security-wise, for the heuristic version of the scheme that we study, we need a q such that $q > 4B$ [7] and the corresponding decisional LWE instance is still hard. Yet, the form of q does not impact the hardness of the problem as shown in [74] and, since q was already extremely close to a power of two for both parameters sets, the practical bit hardness of the corresponding instance is not sensibly changed.

[7] The other condition on q in the parameters table of the submission is to enable the NTT.

Removal of the PRF.

It is well known that in Schnorr-like signatures, a devastating attack is possible if the adversary gets two different signatures using the same y. Indeed, they can simply compute the secret $s = \frac{z-z'}{c-c'}$. While such a situation is very unlikely due to the large size of y, a technique to create a deterministic version of the signature was introduced in [94]. The idea is to compute y as $\mathsf{PRF}(secret\_seed, m)$ such that each message will have a different value for y unless a collision is found in PRF. This modification act as a protection against very weak entropy sources but is not necessary to the security of the signature and was not present in ancestors of qTESLA. Unfortunately, adding this determinism also enabled some side-channel attacks [34, 104]. Hence, the authors of qTESLA decided to take the middle ground by keeping the deterministic design but also seeding the oracle with a fresh random value r.
While those small safety measures certainly make sense if they do not incur a significant performance penalty, we decided to drop it and simply sample y at random at the beginning of the signing procedure. The reason is twofold. First, keeping deterministic generation of y implied masking the hash function evaluation itself which is really inefficient if not needed and would unnecessarily complicate the masking scheme. Second, implementing a masking countermeasure is, in general, making the hypothesis that a reasonable source of randomness (or at least not weak to the point of having a nonce reuse on something as large as y) is available to generate shares and thus can be also used for the signature itself.

4.7.2 Existing Gadgets

First, let us describe gadgets already existing in the literature. Since they are not part of our contribution, we decided to only recall their functionalities without formally describing them.

- SecAnd: Computes the logical and between two values given in boolean masked form, output also in boolean masked form. Order 1 algorithm: [39]. Order n algorithm [19].
- SecAdd: Computes the arithmetic add between two values given in boolean masked form, output also in boolean masked form. Order 1 algorithm: [39]. Order n algorithm [19]. It implicitly reduces the result modulo 2^{word_size}.
- SecArithBoolModq: Converts a value in arithmetic masked form to a value in boolean masked form. Order 1 algorithm: [62]. Order n: [41]. We slightly modify it to an algorithm denoted GenSecArithBoolModq taking into account non power of two number of shares. It can be found in Algorithm 24. When a masked value composed of an odd number of shares t is presented to the algorithm, it first splits them in two uneven parts of size $\lfloor t/2 \rfloor + 1$ and $\lfloor t/2 \rfloor$ before proceeding to the recursive call. The subroutine Expand takes as input an arbitrary number of shares t' and expand them in $2t'$ shares. Applying Expand to both parts, we end up with a part p_1 of size $t + 1$ and a part p_2 of size $t - 1$. We merge the two last shares of p_1 and append a zero to p_2 to get two size t masking that are finally added together to yield the final boolean masking. Note that in practice, the top level call is done from another (non recursive) function that reduces the result in order to have a conversion

modulo q. We recall that thanks to our power of two modulus, this can be done by simply keeping $\log_2 q$ bits of each shares.

- SecBoolArith: Converts a value in boolean masked form to a value in arithmetic masked form. Order 1 algorithm: [62]. Order n algorithm: [38]. This gadget does not explicitly appear in the following but is used inside DataGen.
- DataGen: Takes as input an integer B and outputs a polynomial $y \in \mathcal{R}_{q,[B]}$ in arithmetic masked form. Uses the boolean to arithmetic conversion.
- FullXor: Merges shares of a value in boolean masked form and output the unmasked value.
- FullAdd: Merges shares of a value in arithmetic masked form and output the unmasked value.
- Refresh: Refreshes a boolean sharing using fresh randomness [67].

Algorithm 24 GenSecArithBoolModq

Data: An arithmetic masking $(a_i)_{0\leq i\leq N}$ of some integer x
Result: A boolean masking $(b_i)_{0\leq i\leq N}$ of the same integer x

if $N = 0$ **then**
  $b_0 \leftarrow a_0$
  return $(b_i)_{0\leq i\leq N}$
end if
$\texttt{HALF} \leftarrow \lfloor N/2 \rfloor$
$(x_i)_{0\leq i\leq \texttt{HALF}} \leftarrow \mathsf{GenSecArithBoolModq}((a_i)_{0\leq i\leq \texttt{HALF}})$
$(x'_i)_{0\leq i\leq 2*\texttt{HALF}} \leftarrow \mathsf{Expand}((x_i)_{0\leq i\leq \texttt{HALF}})$
$(y_i)_{0\leq i\leq \lfloor (N-1)/2 \rfloor} \leftarrow \mathsf{GenSecArithBoolModq}((a_i)_{\texttt{HALF}+1\leq i\leq N})$
$(y'_i)_{0\leq i\leq 2*\lfloor (N-1)/2 \rfloor} \leftarrow \mathsf{Expand}((y_i)_{0\leq i\leq \lfloor (N+1)/2 \rfloor})$
if N is even **then**
  $y'_{2*\lfloor (N-1)/2 \rfloor} \leftarrow 0$
  $x'_{2*\texttt{HALF}-1} \leftarrow x'_{2*\texttt{HALF}-1} \oplus x'_{2*\texttt{HALF}}$
end if
$(b_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((x'_i)_{0\leq i\leq N}, (y'_i)_{0\leq i\leq N})$

4.7.3 New Gadgets

To comply with the specifications of qTESLA, our signature scheme includes new components to be masked that were not covered or different than in [19, 92]. In all the following, RADIX refers to the size of the integer datatype used to store the shares.

Absolute value (Algorithm 25): The three checks during the signing procedure are : $z \notin \mathcal{R}_{q,[B-S]}$, $||[w]_L||_\infty \geq 2^{d-1} - E$ and $||w||_\infty \geq \lfloor q/2 \rfloor - E$. They all involve going through individual coefficients (or their low bits) of a polynomial and checking a bound on their absolute value. In the first version of our work, we were actually making two comparisons on each signed coefficients before realizing that it was actually less intensive to explicitly compute the absolute value and do only one comparison. The gadget takes

Algorithm 25 Absolute Value - AbsVal

Data: A boolean masking $(x_i)_{0\leq i\leq N}$ of some integer x and an integer k
Result: A boolean masking $(|x|_i)_{0\leq i\leq N}$ corresponding to the absolute value of $x \bmod^{\pm} 2^k$

1: $(mask_i)_{0\leq i\leq N} \leftarrow ((x_i)_{0\leq i\leq N} << (\text{RADIX} - k)) >> (\text{RADIX} - 1))$
2: $(x'_i)_{0\leq i\leq N} \leftarrow \mathsf{Refresh}((x_i)_{0\leq i\leq N})$
3: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((x'_i)_{0\leq i\leq N}, (mask_i)_{0\leq i\leq N}))$
4: $(|x|_i)_{0\leq i\leq N} \leftarrow ((x_i)_{0\leq i\leq N} \oplus (mask_i)_{0\leq i\leq N}) \wedge (2^k - 1)$

Algorithm 26 Masked rounding - MaskedRound

Data: An arithmetic masking $(a_i)_{0\leq i\leq N}$ of some integer a
Result: An integer r corresponding to the modular rounding of a

1: $(\text{MINUS\_Q\_HALF}_i)_{0\leq i\leq N} \leftarrow (-q/2 - 1, 0, ..., 0)$
2: $(\text{CONST}_i)_{0\leq i\leq N} \leftarrow (2^{d-1} - 1, 0, ..., 0)$
3: $(a'_i)_{0\leq i\leq N} \leftarrow \mathsf{GenSecArithBoolModq}(a_i)_{0\leq i\leq N}$
4: $(b_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((a'_i)_{0\leq i\leq N}, (\text{MINUS\_Q\_HALF}_i)_{0\leq i\leq N})$
5: $b_0 = \neg b_0$
6: $(b_i)_{0\leq i\leq N} \leftarrow ((b_i)_{0\leq i\leq N} >> \text{RADIX} - 1) << \log_2 q$
7: $(a'_i)_{0\leq i\leq N} \leftarrow (a'_i)_{0\leq i\leq N} \oplus (b_i)_{0\leq i\leq N}$
8: $(a'_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((a'_i)_{0\leq i\leq N}, (\text{CONST}_i)_{0\leq i\leq N})$
9: $(a'_i)_{0\leq i\leq N} \leftarrow (a'_i)_{0\leq i\leq N} >> d$
10: **return** $t \coloneqq \mathsf{FullXor}((a'_i)_{0\leq i\leq N})$

as input any integer x masked in boolean form and outputs $|x \bmod^{\pm} 2^k|$. Since computers are performing two's complement arithmetic, the absolute value of x can be computed as follows:

1. $m \leftarrow x >> RADIX - 1$
2. $|x| \leftarrow (x + m) \oplus m$

As we work on signed integers, one can note that the $>>$ in the first step is an arithmetic shift and actually writes the sign bit in the whole register. If x is negative then $m = -1$ (all ones in the register) and if x is positive then $m = 0$. The gadget AbsVal is using the same technique to compute $|x \bmod^{\pm} 2^k|$. The small difference is that the sign bit is in position k instead of position RADIX. This is why line 1 is moving the sign bit (modulo 2^k) in first position before extending it to the whole register to compute the mask.

Masked rounding (Algorithm 26)

In the Bai-Galbraith signature, a compression technique was introduced to reduce the size of the signature. It implies rounding coefficients of a polynomial. Revealing the polynomial before rounding would allow an adversary to get extra information on secret values and thus, this operation has to be done on the masked polynomial. Recall that the operation to compute is $[v]_M = (v \bmod^{\pm} q - [v]_L)/2^d$.
The first step is to compute the centered representative of v, i.e. subtract q from v if $v > q/2$. Taking advantage of our power of two modulus, this operation would be really easy to do if the centered representative was defined as the integer congruent to v in the

Algorithm 27 Masked well-rounded - MaskedWR

Data: Integer $a \in \mathbb{Z}_q$ in arithmetic masked form $(a_i)_{0\leq i\leq N}$
Result: A boolean masking r of $(\|a\| \leq q/2 - E) \wedge (\|[a]_L\| \leq 2^{d-1} - E)$

1: $(\text{SUP\_Q}_i)_{0\leq i\leq N} \leftarrow (-q/2 + E, 0, ..., 0)$
2: $(\text{SUP\_D}_i)_{0\leq i\leq N} \leftarrow (-2^{d-1} + E, 0, ..., 0)$
3: $(a'_i)_{0\leq i\leq N} \leftarrow \mathsf{GenSecArithBoolModq}(a_i)_{0\leq i\leq N}$
4: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{AbsVal}((a'_i)_{0\leq i\leq N}, \log_2 q)$
5: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((x_i)_{0\leq i\leq N}, (\text{SUP\_Q}_i)_{0\leq i\leq N}))$
6: $(b_i)_{0\leq i\leq N} \leftarrow (x_i)_{0\leq i\leq N} >> (\text{RADIX} - 1)$
7: $(a'_i)_{0\leq i\leq N} \leftarrow \mathsf{Refresh}((a'_i)_{0\leq i\leq N})$
8: $(a'_i)_{0\leq i\leq N} \leftarrow (a'_i)_{0\leq i\leq N} \wedge 2^d - 1$
9: $(y_i)_{0\leq i\leq N} \leftarrow \mathsf{AbsVal}((a'_i)_{0\leq i\leq N}, d)$
10: $(y_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((y_i)_{0\leq i\leq N}, (\text{SUP\_D}_i)_{0\leq i\leq N}))$
11: $(b'_i)_{0\leq i\leq N} \leftarrow (y_i)_{0\leq i\leq N} >> (\text{RADIX} - 1)$
12: $(b_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAnd}((b_i)_{0\leq i\leq N}, (b'_i)_{0\leq i\leq N})$
13: **return** $r \coloneqq \mathsf{FullXor}((b_i)_{0\leq i\leq N})$

range $[-q/2, q/2)$ since it would be equivalent to copying the q^{th} bit of v in the most significant part, which can be performed with simple shift operations on shares. Unfortunately, the rounding function of qTESLA works with representatives in $(-q/2, q/2]$. As we wanted compatibility with the original scheme, we decided to stick with their design. Nevertheless, we were still able to exploit our power of two modulus. Indeed, in this context, switching from positive to negative representative modulo q is merely setting all the high bits to one. Hence, we subtract $q/2 + 1$ from v, extract the sign bit b and copy $\neg b$ to all the high bits of v.
The second step is the computation of $(v - [v]_L)/2^d$. We used a small trick here. Subtracting the centered representative modulo 2^d is actually equivalent to the application of a rounding to the closest multiple of 2^d with ties rounded down. Hence we first computed $v + 2^{d-1} - 1$ and dropped the d least significant bits. This is analogous to computing $\lfloor x \rceil = \lfloor x + 0.499\ldots \rfloor$ to find the closest integer to a real value.

Masked well-rounded (Algorithm 27)

Unlike GLP, the signature scheme can fail to verify and may have to be restarted even if the rejection sampling test has been successful. This results from the fact that the signature acts fas a proof of knowledge only on the s part of the secret key and not on the error e. Nonetheless, thanks to rounding, the verifier will be able to feed correct input to the hash function if the commitment is so called 'well-rounded'. Since not well-rounded signatures would leak information on the secret key, this verification has to be performed in masked form.
The MaskedWR gadget has to perform the two checks $||[w]_L||_\infty < 2^{d-1} - E$ and $||w||_\infty < \lfloor q/2 \rfloor - E$. While the cost of this rather simple operation is negligible compared to polynomial multiplication in the unprotected signature, this test is fairly expensive in masked form. Indeed, it requires four comparisons in addition to the extraction of the low bits of w.
After trying the four comparisons method, we realized that the best strategy was actually

Algorithm 28 Rejection Sampling - MaskedRS

Data: A value a to check, in arithmetic masked form $(a_i)_{0\leq i\leq N}$
Result: 1 if $|a| \leq B - S$ else 0

1: $(\text{SUP}_i)_{0\leq i\leq N} \leftarrow (-B + S - 1, 0, ..., 0)$
2: $(a'_i)_{0\leq i\leq N} \leftarrow \mathsf{GenSecArithBoolModq}((a_i)_{0\leq i\leq N})$
3: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{AbsVal}((a'_i)_{0\leq i\leq N}, \log_2 q)$
4: $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((x_i)_{0\leq i\leq N}, (\text{SUP}_i)_{0\leq i\leq N})$
5: $(b_i)_{0\leq i\leq N} \leftarrow ((x_i)_{0\leq i\leq N} >> \text{RADIX} - 1)$
6: **return** $rs \coloneqq \mathsf{FullXor}((b_i)_{0\leq i\leq N})$

to compute both absolute values with the AbsVal gadget. While comparisons only require one SecAdd and one shift, which is less than AbsVal, the cost of all SecAnd operations between the results of those comparisons makes our approach of computing the absolute value slightly better.

Rejection sampling (Algorithm 28)

The rejection sampling procedure consists in ensuring that the absolute value of all coefficients of a polynomial z are smaller than a bound B. In [19], a gadget verifying that the centered representative of a masked integer is greater than $-B$ was applied to both z and $-z$. In [92], a less computationally intensive approach was taken: their rejection sampling gadget takes as input an arithmetic masking of a coefficient $a \in \mathbb{Z}_q$ identified by its canonical representative and check directly that either $a - B$ is negative or $a - q + B$ is positive. This can be easily done using precomputed constants $(-B - 1, 0, ..., 0)$ and $(-q + B, 0, ..., 0)$. Our approach is similar but we use instead the same technique as in the MaskedWR algorithm, that is to first compute the absolute value of a and perform the masked test $||a|| \leq B$. This saves the need for a masked operation to aggregate both tests.

Gaussian Generation (Algorithm 29)

This gadget is needed for the key generation. Following the round 2 specifications of qTESLA, we mask the technique of cumulative distribution table (CDT) used in the key generation. It consists in precomputing a table of the cumulative distribution function of a half Gaussian of standard deviation σ with a certain precision θ. This table contains say T elements p_j for $j \in [0, T]$ such that

$$p_j = 2 \cdot \sum_{i\leq j} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-i^2}{2\sigma^2}} \text{ with } \theta \text{ bits of precision.}$$

The factor 2 is set to consider the half Gaussian. To produce a sample, we generate a random value in $(0, 1]$ with the same precision, and return the index of the last entry in the table that is smaller than that value. We present the masked version of table look up in Algorithm 29. The parameters T and θ are respectively the number of elements of the table and the bit precision of its entries. Note that this algorithm samples only half of the Gaussian, this means that to get a centered distribution, one should sample random signs

for each coefficient afterward. Since, for correctness purpose, the key generation algorithm of qTESLA is checking the sum of the absolute value of the h largest coefficients of each polynomial after Gaussian sampling, we delay the sign choice to later in the key generation.

Algorithm 29 Gaussian Generation - GaussGen

Data: A table of T probability values p_j with θ bits of precision.
Result: An arithmetic masking of an element following a Gaussian of standard deviation σ

initialize $(r_i)_{0\leq i\leq N}$ as a θ-bit Boolean masking of a uniform random value $r \in (0, 1]$
$(x_i)_{0\leq i\leq N} \leftarrow (0, \ldots, 0)$
for $0 \leq j \leq T$ **do**
    initialize $(k_i)_{0\leq i\leq N}$ as a θ-bit Boolean masking of $-p_j$
    $(\delta_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}\big((r_i)_{0\leq i\leq N}, (k_i)_{0\leq i\leq N}\big)$
    $(b_i)_{0\leq i\leq N} \leftarrow (\delta_i)_{0\leq i\leq N} \gg (\theta - 1)$ ▷ 1 when $r < p_j$ or 0 otherwise
    $(b'_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAnd}\,((b_i)_{0\leq i\leq N}, (x_i)_{0\leq i\leq N})$
    initialize $(J_i)_{0\leq i\leq N}$ as a θ-bit Boolean masking of the index j
    $(b_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAnd}\,(\neg(b_i)_{0\leq i\leq N}, (J_i)_{0\leq i\leq N})$ ▷ j when $r \geq p_j$ or 0 otherwise
    $(x_i)_{0\leq i\leq N} \leftarrow (b'_i)_{0\leq i\leq N} \oplus (b_i)_{0\leq i\leq N}$
    $(x_i)_{0\leq i\leq N} \leftarrow \mathsf{Refresh}((x_i)_{0\leq i\leq N})$
end for
return $\mathsf{SecBoolArith}((x_i)_{0\leq i\leq N})$

For the parameters, we take the exact same table as in the reference implementation. This table is kept unmasked because it contains public information. For qTESLA-I, $(T, \theta) = (208, 64)$ and for qTESLA-III, $(T, \theta) = (134, 128)$.

Due to the large size of the table, this process is quite heavy and impacts the efficiency of the key generation. To optimize the size of the table, a Renyi divergence technique is often used. It consists in using an upper bound on the relative error between the CDT distribution and an ideal Gaussian. This upper bound decreases with the maximum number of queries to the algorithm and it is often lower than a statistical distance estimation. Thus, with Renyi divergence techniques, smaller tables allow the same bit security. Interestingly, we could not manage to adapt this method because GaussGen is part of the key generation. First, the maximum amount of queries to the key generation is not clearly bounded in the specifications. Secondly, the security of the key generation is based on a decisional problem (Decisional-RLWE) and it is currently an open problem to apply the Renyi divergence techniques to decisional problems. Another possible optimization to reduce the size of the table would be to use the recursivity provided in [91] (proved with the statistical distance). However, the size of the table will not decrease enough to get an efficient and practical Gaussian sampling.

Masked Check (Algorithm 30)

This gadget is needed for the key generation. Its purpose is to check that the sum of the largest coefficients of the secret key is not too large. To recover the largest coefficients,

this algorithm uses a straightforwardly masked bubble sort by doing h passes on the list of coefficients. The bubble sort uses a masked exchange subroutine where the bit 0 or 1, representing the need for an exchange or not, is also masked. It finishes with a masked comparison with a precomputed bound.

Algorithm 30 Masked Check - MaskedCheck

Data: An arithmetic masking of a polynomial $(s_i)_{0\leq i\leq N}$, a bound S
Result: $ms \coloneqq 1$ if the sum of its h largest coefficients is larger than the bound S and 0 otherwise

1: $(\text{BOUND}_i)_{0\leq i\leq N} = (-S, 0, \ldots, 0)$
2: Find the h largest coefficients $((c_i^{(0)})_{0\leq i\leq N}, \ldots, (c_i^{(h-1)})_{0\leq i\leq N})$ of $(s_i)_{0\leq i\leq N}$ with a masked bubble sort.
3: $(sum_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((c_i^{(0)})_{0\leq i\leq N}, \ldots, (c_i^{(h-1)})_{0\leq i\leq N})$
4: $(\delta_i)_{0\leq i\leq N} \leftarrow \mathsf{SecAdd}((sum_i)_{0\leq i\leq N}, (\text{BOUND}_i)_{0\leq i\leq N})$
5: $(\delta_i)_{0\leq i\leq N} \leftarrow ((\delta_i)_{0\leq i\leq N} >> \text{RADIX} - 1)$
6: **return** $ms \coloneqq \mathsf{FullXor}(\delta)$

Algorithm 31 Masked signature

Data: message m, secret key $sk = ((s_i)_{0\leq i\leq N}, (e_i)_{0\leq i\leq N})$, seed sd
Result: Signature $(z_{unmasked}, c)$

1: $a \leftarrow \mathsf{GenA}(sd)$
2: $(y_i)_{0\leq i\leq N} \leftarrow \mathsf{DataGen}(B)$
3: **for** $i = 0, \ldots, N$ **do**
4: $\quad v_i \leftarrow a \cdot y_i$
5: **end for**
6: $c \leftarrow \mathsf{MaskedHash}((v_i)_{0\leq i\leq N}, m)$
7: $c \leftarrow \mathsf{Encode(c)}$
8: **for** $i = 0, \ldots, N$ **do**
9: $\quad z_i \leftarrow y_i + s_i \cdot c$
10: **end for**
11: **if** $rs \coloneqq \mathsf{FullRS}((z_i)_{0\leq i\leq N}) = 0$ **then**
12: $\quad$ **goto** 2
13: **end if**
14: **for** $i = 0, \ldots, N$ **do**
15: $\quad w_i \leftarrow v_i - e_i \cdot c$
16: **end for**
17: **if** $r \coloneqq \mathsf{FullWR}((w_i)_{0\leq i\leq N}) = 0$ **then**
18: $\quad$ **goto** 2
19: **end if**
20: $z_{unmasked} \leftarrow \mathsf{FullAdd}((z_i)_{0\leq i\leq N})$
21: **return** $(z_{unmasked}, c)$

4.7.4 Masked Scheme

In all signature schemes, two algorithms can leak the secret key through side channels: the key generation algorithm and the signing algorithm.

Masked sign

The masked signature can be found in Algorithm 31. It uses the gadgets described in Section 4.7.3: the gadgets FullRS, FullWR and FullRound denote the extension of MaskedRS, MaskedWR and MaskedRound to all coefficients $j \in [0, n-1]$ of their input polynomial. Beside the removal of the PRF for y, its structure follows closely the unmasked version of the signature.After generating the public parameter a with the original GenA procedure, the gadget DataGen is used to get polynomials y_i such that $y = \sum_{i=0}^{N} y_i$ belongs to $\mathcal{R}_{q,[B]}$. Then, thanks to the distributive property of the multiplication of ring elements, we can compute $v = a \cdot y = \sum_{i=0}^{N} a \cdot y_i$ using regular polynomial multiplication, without relying on any complex gadget. The polynomial c is computed using the subroutine MaskedHash which is using the MaskedRounding gadget to compute qTESLA's rounding and hashing on a masked polynomial. Once c has been computed, the candidate signature can be computed directly on shares with the masked secret key as $z = y + s \cdot c = \sum_{i=0}^{N} y_i + s_i \cdot c$. Writing FullRS and FullWR to denote the extension of the MaskedRS and MaskedWR gadgets to all the coefficients of a polynomial, the security and correctness parts of the signature follow trivially. Once all checks have been passed, the signature can be safely unmasked using FullAdd and the signature output.

UF-CMA security of the masked scheme

In Theorem 1, we will prove that one iteration of the signature is secure in the probing model. This means that the model assumes that the adversary will only see the leakage of one iteration of the signature. Indeed, from a practical point of view, since the masked key is the same at each iteration, the adversary can thus attack the individual shares s_i using DPA over multiple signatures. This is unsatisfactory since it gives an easy attack in practice and the common security model UF-CMA allows the adversary to make multiple queries to a signing oracle before outputting a forgery. This issue in discussed in [19] with an extention of the UF-CMA game to the probing model. The solution to get security with multiple queries (and thus in practice, the observation of multiple signatures) is to introduce a key update algorithm that refreshes the key shares and that should be run after each signature.

Algorithm 32 MaskedHash

Data: The n coefficients $a^{(j)}$ to hash, in arithmetic masked form $(a_i^{(j)})_{0 \le i \le N}$ and the message to sign m
Result: Hash of the polynomial c

1: Let t be a byte array of size n
2: **for** $j = 1$ to n **do**
3: $\quad t_j \leftarrow \mathsf{MaskedRound}((a_i^{(j)})_{0 \le i \le N})$
4: **end for**
5: $c \leftarrow \mathsf{H}(t, m)$
6: **return**

Masked key generation

As the number of signature queries per private key can be high (up to 2^{64} as required by the NIST competition), whereas the key generation algorithm is typically only executed once per private key, the vulnerability of the key generation to side channel attacks is therefore less critical. We nevertheless present an (inefficient) masked version of the key generation algorithm that can be found in Algorithm 33. One can remark that the bottleneck gadget, GaussGen, needs to make T comparisons for each coefficient of the polynomial which goes to a total of $T \cdot n$ comparisons for the whole generation. With a value of T around 200, sampling from the table is actually sensibly heavier than signing. Thus, our goal with this masked QTESLA key generation is to prove that masking without changing the design is costly but still doable. On the bright side, the Gaussian generation of QTESLA, even masked, is already quite efficient at the cost of storing the table [20]. However, for a practical implementation in which the key generation might be vulnerable to side channels, one could prefer changing the design of the scheme. For example, DILITHIUM generates the keys uniformly at random on a small interval and thus avoids this issue. One downside of this faster key generation is that the parameters of the scheme should be adapted in order to avoid having to much rejections in the signing algorithm.

In Algorithm 33, the element a is generated in unmasked form because it is also part of the public key. Then, s and e are drawn using the gadget GaussGen introduced in Section 4.7.3. Another gadget FullCheck is also introduced in the key generation. It checks that the sum of the h largest entries (in absolute value) is not above some bounds that can be found in Table 4.1. Then the public key t is computed in masked form and securely unmasked with the FullAdd gadget.

Algorithm 33 Masked key generation

Result: Secret key $sk = ((s_i)_{0\leq i\leq N}, (e_i)_{0\leq i\leq N}, sd)$, public key $pk = (sd, t)$

pre-seed $\xleftarrow{r} \{0,1\}^\kappa$
sd $\leftarrow$ PRF(pre-seed)
$a \leftarrow \mathsf{GenA}(sd)$
do
  $(s_i)_{0\leq i\leq N} \leftarrow \mathsf{GaussGen}()$
while (MaskedCheck$((s_i)_{0\leq i\leq N}, S) \neq 0$)
do
  $(e_i)_{0\leq i\leq N} \leftarrow \mathsf{GaussGen}()$
while (MaskedCheck$((e_i)_{0\leq i\leq N}, E) \neq 0$)
initialize $(sign_i^s)_{0\leq i\leq N}$ and $(sign_i^e)_{0\leq i\leq N}$ as two 1-bit arithmetic masking of either -1 or 1
$(s_i)_{0\leq i\leq N} \leftarrow$ SecAnd $((sign_i^s)_{0\leq i\leq N}, (s_i)_{0\leq i\leq N})$
$(e_i)_{0\leq i\leq N} \leftarrow$ SecAnd $((sign_i^e)_{0\leq i\leq N}, (e_i)_{0\leq i\leq N})$
$(t_i)_{0\leq i\leq N} \leftarrow a \cdot (s_i)_{0\leq i\leq N} + (e_i)_{0\leq i\leq N} \bmod q$
$t \leftarrow \mathsf{FullAdd}((t_i)_{0\leq i\leq N})$
$sk \leftarrow ((s_i)_{0\leq i\leq N}, (e_i)_{0\leq i\leq N}, sd)$
$pk \leftarrow (sd, t)$
return sk, pk

4.7.5 Proof of Masking

In Section 4.5, we give the intuition that masking was helping again side-channel attacks. However, to formally analyze this countermeasure, we need to somehow model the capabilities of the attacker. Of course, it is hopeless to try to prevent all type of side-channel attacks since these are somewhat arbitrary powerful but the goal is to provide security for a large class of them. The model we use is called the *probing model* [67]. Basically, a cryptographic implementation is N-probing secure if any set of at most N intermediate variables is statistically independent of the secrets. It thus models an adversary that can perform observations on $\delta \leq N$ arbitrary values during the execution of the algorithm. Naturally, a randomly masked (at order N) value is secure in the probing model but some complications appear when considering gadgets that manipulates multiple shares. Here, the concept of observation is defined in terms of probing a wire in a circuit implementing the cryptographic scheme.

Proofs by composition

To achieve N-probing security, Barthe et al. formally defined two security properties in [18], namely *non-interference* and *strong non-interference*, which ease the security proofs for small gadgets and allows to securely combine secure gadgets together.

Definition 40. *A gadget is N-non-interfering (N-NI) iff any set of at most N observations can be perfectly simulated from at most N shares of each input.*

Definition 41. *A gadget is N-strong non-interfering (N-SNI) iff any set of at most N observations whose N_{int} observations on the internal data and N_{out} observations on the outputs can be perfectly simulated from at most N_{int} shares of each input.*

It is easy to check that N-SNI implies N-NI which implies N-probing security. The strong non-interference only appears in the proofs for subgadgets inside the signature and key generation algorithm. An additional notion was introduced in [19] to reason on the security of lattice-based schemes in which some intermediate variables may be revealed to the adversary.

Definition 42. *A gadget with public outputs X is N-non-interfering with public outputs (N-NIo) iff every set of at most N intermediate variables can be perfectly simulated with the public outputs and at most N shares of each input.*

In the full version of our work [60], we proved some security properties of our new gadgets of Section 4.7.3. Since this part of the work was not done by the author of this thesis, results are given in Table 4.2 and proofs are omitted. However, to exemplify the concept, we give the proof of the main masking theorem stating the security of the composition used to compute the signature. For simplicity and without losing generality, the theorem only considers one iteration for the signature: the signing algorithm outputs $\perp$ if one of the tests in steps 12 or 18 in Algorithm 31 has failed. We also assume the security properties of Table 4.2. We denote by $\left(r^{(j)}\right)_{0\leq j<n}$,$\left(rs^{(j)}\right)_{0\leq j<n}$ and $\left(u^{(j)}\right)_{0\leq j<n}$ the outputs of FullRS, FullWR and FullRound (the values for each coefficient $j \in [0, n-1]$).

Existing Gadgets			New Gadgets	
Name	Property	Reference	Name	Property
SecAnd	N-NI	[39], [19]	GenSecArithBoolModq	N-NI
SecAdd	N-NI	[39], [19]	AbsVal	N-NI
SecABModq	N-SNI	[62], [41]	MaskedRound	N-NIo
SecBoolArith	N-NI	[62], [41]	FullRound	N-NIo
FullXor	N-NIo	[19]	MaskedWR	N-NIo
FullAdd	N-NIo	[19]	FullWR	N-NIo
DataGen	N-NIo	[19]	MaskedRS	N-NIo
MultAdd	N-NI	[19], denoted H^1	FullRS	N-NIo
Refresh	N-SNI	[67]	GaussGen	N-NI
			MaskedCheck	N-NIo

Table 4.2: Security properties of the known and new gadgets.

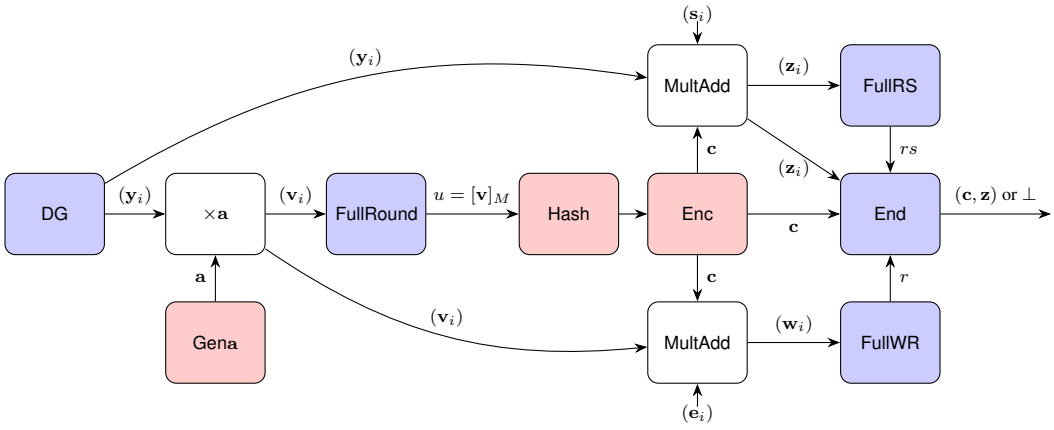

Figure 4.7: Masked Signature structure (The white (resp. blue, red) gadgets are proved N-NI (resp. N-NIo, unmasked)). The non sensitive element sd is omitted for clarity.

Theorem 1. *Each iteration of the masked signature in Algorithm 31 is N-NIo secure with public outputs*[8]

$$\left\{ \left(r^{(j)}\right)_{0\leq j<n}, \left(rs^{(j)}\right)_{0\leq j<n}, \left(u^{(j)}\right)_{0\leq j<n} \right\}$$

(and the signature if returned).

Proof. The overall gadget decomposition of the signature is in Figure 4.7.

Gadgets. The gadget $\times a$ multiplies each share of the polynomial y by the public value a. By linearity, it is N-NI. The gadget FullRound denotes the extension of the MaskedRound to all coefficients of v and is N-NIo. The gadget MultAdd takes $(y_i)_{0\leq i\leq N}$, $(s_i)_{0\leq i\leq N}$ and c (resp. $(v_i)_{0\leq i\leq N}$, $(e_i)_{0\leq i\leq N}$ and c) and computes $(z_i)_{0\leq i\leq N} = (y_i)_{0\leq i\leq N} - c \cdot (s_i)_{0\leq i\leq N}$ (resp. $(w_i)_{0\leq i\leq N} = (v_i)_{0\leq i\leq N} - c(e_i)_{0\leq i\leq N}$). The gadget End simply outputs

[8]Here also, the number of iterations of the gadget DataGen is omitted as a public output.

$(\mathsf{FullAdd}((\mathbf{z}_i)_{0\leq i\leq N}), c)$ if rs and r are true; and $\perp$ otherwise. By the N-NIo security of FullAdd, this gadget is also N-NIo secure.

Thus, all the subgadgets involved are either N-NI secure, N-SNI secure, N-NIo secure or they do not manipulate sensitive data. We prove that the final composition of all gadgets is N-NIo. We assume that an attacker has access to $\delta \leq N$ observations. Our goal is to prove that all these δ observations can be perfectly simulated with at most δ shares of $(s_i)_{0\leq i\leq N}$ and $(e_i)_{0\leq i\leq N}$ and the knowledge of the outputs.
In the following, we consider the following distribution of the attacker's δ observations:

- δ_1 observed during the computations of DG that produces shares of $(y_i)_{0\leq i\leq N}$,
- δ_2 observed during the computations of the gadget $\times a$ that produces the shares of $(v_i)_{0\leq i\leq N}$,
- δ_3 observed during the computations of FullRound,
- δ_4 observed during the computations of the upper MultAdd gadget that produces $(z_i)_{0\leq i\leq N}$,
- δ_5 observed during the computations of the lower MultAdd gadget that produces $(w_i)_{0\leq i\leq N}$,
- δ_6 observed during the FullRS,
- δ_7 observed during the FullWR,
- δ_8 observed during the End.

Some observations may be done on the unmasked gadgets (GenA, Hash and Enc) but their amount will not matter during the proof. Finally, we have $\sum_{i=1}^{8} \delta_i \leq \delta$.

We build the proof from right to left. The gadgets End, FullRS, FullRound and FullWR are N-NIo secure with the output (z, c) or $\perp$ (resp. $\left(rs^{(j)}\right)_{0\leq j<n}$, $\left(u^{(j)}\right)_{0\leq j<n}$, $\left(r^{(j)}\right)_{0\leq j<n}$). As a consequence, all the observations from their call can be perfectly simulated with at most δ_8 (resp. δ_6, δ_7) shares of z (resp. z, w). For the upper MultAdd gadget, there are at most $\delta_8 + \delta_6$ observations on the outputs and δ_4 local observations. The total is still lower than δ and thus they can be simulated with at most $\delta_4 + \delta_6 + \delta_8 \leq \delta$ shares of y and s.
Concerning the lower MultAdd gadget, there are at most δ_7 observations on w and δ_5 made locally. Thus they can be simulated with at most $\delta_5 + \delta_7 \leq \delta$ shares of v and e.
The gadget FullRound is N-NIo so all the observations from its call can be simulated with at most δ_3 shares of v. Thus, there are $\delta_3 + \delta_5 + \delta_7$ observations on the output of gadget $\times a$. And then, they can be simulated with at most $\delta_3 + \delta_5 + \delta_7 + \delta_2$ shares of y. Summing up all the observations of y gives $(\delta_3 + \delta_5 + \delta_7 + \delta_2) + (\delta_4 + \delta_6 + \delta_8) \leq \delta$. This allows to conclude the proof by applying the N-NIo security of DG. All the observations on the algorithm can be perfectly simulated with at most $\delta_4 + \delta_6 + \delta_8 \leq \delta$ shares of s, $\delta_5 + \delta_7 \leq \delta$ shares of e and the knowledge of the public outputs.

□

4.7.6 Practical Aspects and Implementation Details

Our masking scheme has been implemented inside the reference code of qTESLA available on the repository of their project [108].

We added two new files called `base_gadgets.c` and `sign_gadgets.c` containing all the algorithms manipulating masked values. The actual masked signature (Algorithm 31) is available in `sign.c`. Beside, some modifications related to the new modulus have been made in various places but the overall structure of the code is the same as before. The random oracle of the signature is implemented with `cSHAKE`.

Randomness

The generation of random numbers plays an important role in the performances of the scheme since most of the basic gadgets need fresh randomness in the form of unsigned 32-bit integers. Our function retrieving randomness is called `rand_uint32()`. It is defined as a macro in `params.h` in order to easily be disabled for testing purpose. Our tests with the randomness enabled were performed using xoshiro128** [24], a really fast PRNG that has been recently used to speed-up public parameters generation in a lattice-based cryptosystem [30]. One looking for real life application of our technique and believing masking need strong randomness would maybe want to use a cryptographically secure PRNG instead. Another option could be to expand a seed with the already available `cSHAKE` function but as we will see in the next section, it might be pretty expensive as the number of random bytes required grows very fast with the number of shares.

Performances

We benchmarked our code on a laptop with a CPU Intel Core i7-6700HQ running at 2.60GHz as well as on a Cortex-M4 microcontroller for the masking of order 1. We emphasis that unlike the work presented in the previous chapter, the code was not optimized for the Cortex-M4, we directly compiled our reference C code without modification.The result for individual gadgets over 1 000 000 executions can be found in Table 4.3. The table is divided in two parts: the top part contains measurements for the signing gadgets implementing functionalities of the signature and the bottom part contains measurements for the base gadgets implementing elementary operations. Unsurprisingly, we see that the most expensive signing gadget is MaskedWR. Indeed, it has to perform two absolute value computations in addition to two comparisons. Nevertheless, an actual substantial overall gain of performances would rather come from an improvement of the conversion from arithmetic to boolean masking since it the slowest base gadget and is used in all signing gadgets. Furthermore, it should be also pointed out that most gadgets have a non negligible dependency on the speed of SecAnd since it is called multiple times in SecAdd which itself appears multiple times in signing gadgets. The results for the full signature are given in Table 4.4 and Table 4.5. Asymptotically, the complexity of the scheme will be driven by the most expensive gadget, GenSecArithBoolModQ which runs in $\mathcal{O}(N^2 \cdot \log \text{RADIX})$ [40]. However, we believe that for the low orders that might realistically be used in practice, asymptotic complexity is a somewhat irrelevant metric of efficiency. Actually, during informal testings, we observed that for some orders, a gadget was faster than one of its lower complexity equivalent because it needed less randomness. Since a large portion of the execution time is spent in calls to the random number generator, we decided to benchmark with and without the PRNG. The mention

Masking order	Order 1	Order 2	Order 3	Order 4	Order 5
RG	98	410	840	1 328	2 416
MaskedRound	164	1 400	2 454	4 314	6 142
MaskedWR	280	2 080	3 914	6 432	9 034
MaskedRS	178	1 440	2 496	4 432	6 254
SecAdd	44	294	592	870	1 192
SecAnd	20	28	44	70	96
GenSecArith-BoolModQ	96	786	1 152	3 148	3 500
SecBoolArith	20	42	108	288	884

Table 4.3: Median speed of principal gadgets in clock cycles over 1000000 executions

RNG off means that `rand_uint32()` was set to return 0. The mention RNG on means that `rand_uint32()` was set to return the next value of xoshiro128**. The purpose is to give an idea of how the algorithm itself is scaling, regardless of the speed at which the device is able to provide randomness. At the same time, the discrepancy between the values with and without the RNG underlines how masking schemes of this magnitude are sensitive to randomness sampling. In Table 4.7, we also computed the average number of calls to `rand_uint32()` to see how much randomness is needed for each order. Each call is retrieving a uniformly random 32-bit integer. As expected, this number is growing fast when the masking order is increased. The results for the masked signature at order 1 on Cortex-M4 microcontroller are given in Table 4.6. We speculate that the scaling difference between the microcontroller and the computer is due to the fact that architectural differences matter less for the masking code than for the base signature code. Furthermore, we can see that qTESLA-III is scaling better than qTESLA-I. Beside the natural variance of the experiments, we explain this result by the fact that increasing the masking order reduces the impact of the polynomial multiplication on the timing of the whole signature in favor of masking operations. Factoring out polynomial operations, qTESLA-III is scaling better because the probability of rejection for this parameters set is lower than for qTESLA-I. Hence, even if n is twice as large, less than twice the masking operations are performed overall.

As noted in [92], the power of two modulus allows to get a reasonable penalty factor for low masking orders. Without such a modification, the scheme would have been way slower. Besides, our implementation seems to outperform the masked implementation of DILITHIUM as given in [92]. The timing of our order 1 masking for qTESLA-I is around 1.3 ms, and our order 2 is around 7.1 ms. This result comes with no surprise because the unmasked version of qTESLA already outperformed DILITHIUM. However, we do not know if our optimizations on the gadgets could lead to a better performance for a masked DILITHIUM.

Masking order	Unmasked	Order 1	Order 2	Order 3	Order 4	Order 5
qTESLA-I (RNG off)	645 673	2 394 085	7 000 117	9 219 826	16 577 823	24 375 359
qTESLA-I (RNG on)	671 169	2 504 204	13 878 830	24 582 943	39 967 191	59 551 027
qTESLA-I (RNG on) Scaling	1	$\times 4$	$\times 21$	$\times 37$	$\times 60$	$\times 89$

Table 4.4: Median speed of masked signature in clock cycles over 10000 executions for qTESLA-I on Intel Core i7-6700HQ running at 2.60GHz

Masking order	Unmasked	Order 1	Order 2	Order 3	Order 4	Order 5
qTESLA-III (RNG off)	1 252 645	4 511 179	9 941 571	14 484 664	25 351 066	34 415 499
qTESLA-III (RNG on)	1 318 868	4 138 907	21 932 379	33 520 922	59 668 280	83 289 124
qTESLA-III (RNG on) Scaling	1	$\times 3$	$\times 17$	$\times 25$	$\times 45$	$\times 63$

Table 4.5: Median speed of masked signature in clock cycles over 10000 executions qTESLA-III

4.8 Conclusion

In this chapter, we described the two faces of side-channel: attacks and countermeasures. First, we provided experimental results of a CPA attack performed on a new block cipher called Kalyna. The expected success of our attack confirmed that even if the cipher is different, the threat of side-channel attacks will still be present if the designer do not take them in consideration during design by facilitating the implementation of countermeasures. Second and more importantly, we provided a masking scheme on the NIST lattice-based signature qTESLA. This work is part of a common effort from the community to study different aspects of NIST's post-quantum competition candidates. This is especially relevant to the second round of the process in which practical aspects must be studied more closely. While the masking of qTESLA is naturally similar to other Fiat-Shamir lattice-based signatures, some specificities had to be taken into consideration in order to get a fully masked scheme. Unlike previous work, we used state-of-the-art algorithms for all the gadgets and specialized ones for masking of order 1. Furthermore, thanks to small modifications to the scheme itself, namely the removal of the PRF and the usage of a power of two modulus, the cost of masking is reasonable, at least for small orders. This indicates that some design elements that seem to be a good idea for the unprotected scheme might be actually problematic in practice. We backed up these claims by providing benchmarks with a C implementation inside the original code of the designers of the scheme.

4.9 Thoughts and Future Works

The CPA on Kalyna was pretty straightforward and unsurprisingly gave good results. Even though attacking the round keys was a bit more involved than retrieving a master key, there is no reason that a straightforward implementation of such a cipher would

Masking order	Unmasked	Order 1
qTESLA-I Cortex-M4	11 304 025	23 519 583

Table 4.6: Median speed of masked signature in clock cycles over 1000 executions for qTESLA-I on Cortex-M4 microcontroller

Masking order	Order 1	Order 2	Order 3	Order 4	Order 5
qTESLA-I	85 810	1 383 459	2 761 525	4 923 709	7 638 422
qTESLA-III	115 392	1 826 545	3 721 800	6 482 130	10 005 714

Table 4.7: Average number of calls to `rand_uint32()`

resist basic DPA. Yet, it helped me discovering the topic of side-channel attacks and was the very first scientific paper I was involved in. It was also interesting for me to see that these attacks are doable in practice and do not only exist in textbooks. On the other hand, I think the paper on qTESLA is very relevant. In the framework of the NIST standardization project, a lot of effort has been recently put toward practicality and finding countermeasures against side-channel attacks is part of it. My opinion is that we are now aware that this type of attack exists and thus, must be discussed before standardization. For example, to mask the lattice-based schemes studied in this thesis, it seems clear that choosing q as a power of two is the preferred choice. However, most schemes use $q \equiv 1 \mod 2n$ to enable fast polynomial multiplication with the NTT. This raises the following question: do we care more about efficiency for unprotected implementation than for masked implementation? This question might we tricky because the two points of view

- I use a prime q because I do not always need side-channel protections and when I do, I accept a huge overhead, and;
- I use a power of two q because Karatsuba/Toom-Cook is still pretty fast and is a good tradeoff for side-channel countermeasure;

both make sense depending on the context. I personally do not have a strong opinion on that but I feel like this issue is mostly ignored in the process right now and since some things might have to be modified in the design of the primitives themselves, it will be too late in the later rounds of the project.

Regarding future works, there are two directions that interest me. First, I would like to compare more closely the performances of masked Dilithium and qTESLA with implementation optimized for embedded devices. I feel like the speed results from the masking of Dilithium published in [92] might be improved by applying some of our techniques. Second, I am interested in finding a design for lattice-based Fiat-Shamir signatures that is as masking friendly as possible. This was actually the goal when I started working with Mélissa Rossi but we dropped the idea because the community was mostly interested by NIST candidates and not new proposals. However, I am still curious and I think a design à la GLP with large parameters and low rejection probability might give good scaling results.

Chapter 5

Efficient Design for Lattice-Based Cryptography

5.1 Preamble

While being the last, this chapter discuss the first research paper I initiated and conducted to publication by myself. It is authored by me and my non-cryptographer friend Keno Merckx who helped me bound the failure probability of the scheme. The reason why I put this work last is that the scheme itself is built upon lattice-based signature and encryption and thus the flow of the document is better with Chapter 3 and Chapter 4 previously introducing relevant material. The topic of this work is the design of a signcryption scheme based on lattice assumptions. Since there exists a variant of the famous signcryption scheme of Zheng based on Schnorr signature, the idea to transpose it to the lattice setting came pretty fast to my mind. However, several iterations were needed to reach a satisfactory result. An early version of the scheme based on the GLP signature was proposed but rejected due the oldness of the underlying scheme. After several modifications, a version based on the scheme of Bai-Galbraith/Tesla sharp was accepted to CANS 2018 [59]. Since lattice signatures and KEMs evolved in the past two years due to the ongoing NIST competition, some parts might feel slightly outdated but the concept of the scheme still stands.

5.2 Introduction

The two last chapters were studying two practical issues of existing schemes, namely optimized implementations and resistance to physical attacks. Nevertheless, another way to gain efficiency is to have a scheme which is streamlined for the needs of the user. Indeed, cryptographic schemes are mainly designed as black boxes which have a very specific task. The user setting up a secure environment will pick several of this boxes and use them one after the other to reach all the security requirements. The advantage of this approach is that cryptographers only have to focus on the security of the boxes and users subsequently have the responsibility to use them properly. But sometimes, some properties covered by different type of primitives are so often needed together that creating a specific one covering all of them makes sense.

This is especially true for the most natural security properties that are confidentiality, data integrity and authentication. In a public-key setting, they are ensured using a PKE

and a signature scheme. Those cryptographic primitives have been developed somewhat independently and can be used separately, depending on the context. If the adversary is passive, i.e they can only read the channel but not write on it, encryption can be enough. If the secrecy of the message is not important, signing can be enough. Yet, in a situation in which an active adversary is present during a sensitive communication, confidentiality, data integrity and authentication must all be guaranteed at the same time. It is clearly possible to use encryption and signature together but it implies accepting the overhead of using two building blocks and forces a careful security analysis since concatenating two cryptographic primitives in a naive way can be dangerous, for example, revealing the signature on a message will often endanger its confidentiality.

In symmetric cryptography, a lot of effort has been put toward the development of *authenticated encryption* schemes. The idea is to merge a symmetric encryption scheme with a message authentication code in a single block providing all the security properties listed above. This work gave rise to a dedicated workshop (DIAC) and a competition to establish a portfolio called CAESAR.

On the public-key side, the equivalent primitive is called *signcryption*. The goal of a signcryption scheme is to provide the security properties of both encryption and signature at a lower cost than concatenating them. The (academic) story started at CRYPTO in 1997 with the original paper of Zheng [124]. In this work, the author used a clever combination of ElGamal encryption and signature to create an efficient scheme leading a line of research aiming at formalizing, studying security and enhancing signcryption [45]. Unfortunately, the techniques used were based on the Diffie-Hellman (or RSA) assumption and their security would be compromised in case of the emergence of a large quantum computing power.

In the following sections, we introduce a construction of a signcryption scheme in the Fiat-Shamir with aborts framework of Lyubashevsky based on the signature of Bai and Galbraith [15]. It is inspired from a Schnorr-like variant of the original work of Zheng [124] proposed by Malone-Lee [86]. We provide two versions of the scheme, both relying on the concept of sharing a key while signing and forwarding a symmetric encryption of the message under this key. The first one uses a usual lattice-based key exchange while the second one encrypts the key in a KEM (key encapsulation mechanism) fashion. Those two flavors of the scheme provide a tradeoff between efficiency and storage. The key exchange version is slower but uses less memory/bandwidth. We also provide a concrete instantiation with parameters chosen according to the methodology of [15] enabling correctness of the scheme and compares the gains of using this specific scheme instead of a naive concatenation of signature and key exchange.

Signcryption has not been extensively studied in the post-quantum world yet. Some works on lattice-based schemes exist [77, 79, 110, 122], however, they are all based on trapdoors and thus provide less practical instantiations. Our work was, at the time of publication, the first one studying signcryption using the Fiat-Shamir with aborts technique on lattices. We called the scheme SETLA (Signature and EncrypTion from LAttices) as a tribute to the TESLA family of signatures and to facilitate references to it in the text.

5.3 Signcryption

A signcryption scheme is a cryptographic primitive aiming to act at the same time as encryption and signature on some data. The usual situation is that of a sender (a.k.a Alice)

willing to send a message m to a receiver (a.k.a Bob) while ensuring at the same time confidentiality, integrity and authentication. It is the public-key analog of authenticated encryption.

Definition 43. *Formally, a signcryption scheme with message space $\mathcal{M}$ and signcryptext space $\mathcal{C}$ is a tuple $\Gamma_{\mathcal{M},\mathcal{C}} = (\texttt{ParamGen}, \texttt{KeyGenSender}, \texttt{KeyGenReceiver}, \texttt{Signcrypt}, \texttt{Unsigncrypt})$ composed of the five following algorithms:*

- $\texttt{ParamGen}(\lambda)$*: a randomized algorithm taking as input the security parameter λ and outputting the parameters* `params` *of the system. We consider* `params` *as an implicit input of all the algorithms.*
- $\texttt{KeyGenSender}()$*: a randomized algorithm generating a key pair (sk_a, pk_a) for the sender (Alice). We will call sk_a the secret signing key and pk_a the public verification key.*
- $\texttt{KeyGenReceiver}()$*: a randomized algorithm generating a key pair (sk_b, pk_b) for the receiver (Bob). We will call sk_b the secret decryption key and pk_b the public encryption key.*
- $\texttt{Signcrypt}(sk_a, pk_b, m)$*: a randomized algorithm taking as input Alice's secret signing key sk_a, Bob's public encryption key pk_b, a message $m \in \mathcal{M}$ and outputting a signcryptext $C \in \mathcal{C}$.*
- $\texttt{Unsigncrypt}(pk_a, sk_b, C)$*: a deterministic algorithm taking as input Alice's public verification key pk_a, Bob's secret decryption key sk_b, a signcryptext $C \in \mathcal{C}$ and outputting a either a message $m \in \mathcal{M}$ if the signcryptext is valid or a failure symbol $\perp$.*

It should be noted that, for efficiency and simplicity reasons, the two key generation algorithms can be merged in a single `KeyGen` *algorithm outputting a key pair (sk, pk) in which sk act simultaneously as decryption and signing key and pk as verification and encryption key.*
The natural correctness property is that, for every valid m,

$$\texttt{Unsigncrypt}(pk_a, sk_b, \texttt{Signcrypt}(sk_a, pk_b, m)) = m$$

with an overwhelming probability.

Non-repudiation. There is no settled answer to the question of non-repudiation for a signcryption scheme. Indeed, since we want confidentiality of the message, it is not clear if a public verification mechanism is required. But if Alice can later repudiate the message in front of a judge, can we really call it a signature? The consensus is to set up a mechanism allowing Bob to generate a signature from the signcryptext at the price of revealing the message. Hence, if at some point Alice tries to be dishonest, he can create a publicly verifiable signature and present it to the judge. Hence, we extend the signcryption scheme with two optional algorithms:

- $\texttt{SignExtract}(pk_a, sk_b, C)$: a deterministic algorithm taking the same inputs as `Unsigncrypt` and outputting a publicly verifiable signature $\sigma(m)$.

- `PublicVerif`$(pk_a, \sigma(m))$: a deterministic algorithm taking as input the parameters of the system, the public key of Alice and a signature on m and outputting 1 if $\sigma(m)$ is a valid signature on m, 0 otherwise.

In practice, the `SignExtract` algorithm can be merged with `Unsigncrypt` to output at the same time m together with its signature $\sigma(m)$.

5.4 Security model for signcryption

We naturally extend the security games of encryption and signature to the case of signcryption. As an example, here follow the usual IND-CPA and sUF-CMA games, which are analogous to Definition 7 and Definition 14:

Definition 44. *The signcryption scheme is said to be IND-CPA secure if the probability of an adversary (having a set of two PPT algorithms $\mathcal{A}, \mathcal{A}'$) winning the following game is negligibly close to $\frac{1}{2}$*

1. The challenger first runs the `ParamGen` algorithm and outputs public parameters `params`. After that, the challenger generates Alice's key pair $(pk_a, sk_a) \leftarrow$ `KeyGenReceiver`$()$ and Bob's pair $(pk_b, sk_b) \leftarrow$ `KeyGenReceiver`$()$.
2. The adversary runs $\mathcal{A}$ on input (pk_a, pk_b). It has access to a `Signcrypt`(sk_a, pk_b, m) and an `Unsigncrypt`(pk_a, sk_b, C) oracle. The algorithm finishes by outputting two messages m_0 and m_1.
3. The challenger chooses a bit $b \xleftarrow{r} \{0, 1\}$ and outputs $\hat{C} \leftarrow$ `Signcrypt`(sk_a, pk_b, m_b)
4. The adversary then runs $\mathcal{A}'$ on input $(\hat{C}, pk_a, sk_a)$ and finishes by outputting a bit b'.

The adversary wins the IND-CPA game if $b' = b$.

Definition 45. *The signcryption scheme is said to be sUF-CMA secure if the probability of the adversary (having a PPT algorithm $\mathcal{A}$) winning the following game is negligible.*

1. The challenger first runs the `ParamGen` algorithm and outputs public parameters `params`. After that, the challenger generates Alice's key pair $(pk_a, sk_a) \leftarrow$ `KeyGenReceiver`$()$ and Bob's pair $(pk_b, sk_b) \leftarrow$ `KeyGenReceiver`$()$.
2. The adversary runs $\mathcal{A}$ on input (pk_a, pk_b). It has access to a `Signcrypt`(sk_a, pk_b, m) and an `Unsigncrypt`(pk_a, sk_b, C) oracle. The algorithm finishes by outputting a signcryptext $\hat{C}$.

The adversary wins the sUF-CMA game if $\hat{C}$ has not been output by the `Signcrypt` oracle and `Unsigncrypt`$(pk_a, sk_b, \hat{C}) \neq \perp$.

Finding the right security model for signcryption seems to be a bit more complex than for other basic primitives. Indeed, since for every communication each participant plays a dual role using at the same time is own secret and the public value of the other one, it is unclear what power we should give to the adversary. To clarify the situation, signcryption schemes security is defined according to two notions, insider security and outsider security.

Signing key: $sk \xleftarrow{r} \mathbb{Z}_q$
Verification key: $pk \leftarrow g^{sk}$

ElGamal sign(sk, m):

1: $r \xleftarrow{r} \mathbb{Z}_q$
2: $c \leftarrow \mathsf{H}(m||g^r)$
3: $z \leftarrow r \cdot (sk + c)^{-1}$
4: **return** c, z

ElGamal Verify(c, z, pk):

1: $\omega \leftarrow (pk \circ g^c)^z (= g^r)$
2: **return** $c = \mathsf{H}(m||\omega)$

Figure 5.1: Variant of ElGamal Signature

Outsider security

In the outsider model, the sender and the receiver are both honest and try to prevent an external adversary from retrieving information about the message or modifying it without being detected. It is the model used in the symmetric case of authenticated encryption where all the valid users are "the same" in the sense that they all have the same power and are not associated to a public identity. Here, the adversary only has access to the public-key of the different users of the system and tries to break the IND-CPA or sUF-CMA property of the scheme. A signcryption scheme insecure in the outsider model would be of no use.

Insider security

The insider security model is more complete but its usefulness in practice is more debatable. In this one, the receiver and the sender can be the adversary (a more realistic view is that an adversary is given the private key of one of them). Concerning the dishonest receiver, his goal is to forge a signcryptext (signcrypted with his own public key) on a new message without knowing the sender's secret key. For the dishonest sender, her goal is to unsigncrypt a signcryptext created with her private key. The pertinence of this model is quite context dependent. If Bob is the only entity capable of verifying the authenticity of a message, his power to forge a message for himself is irrelevant. Likewise, if Alice can decrypt signcryptext she created with her signing key, we can fairly consider she knows the message. The argument against this claim is to try to protect past (or future) communication if Alice's private key is somehow compromised. In our case, we will not allow Bob to forge a message because we want to enable non-repudiation, meaning that he is not the only one capable verifying the signature anymore. On the other side, we allow Alice to be able to recover a message from past signcryption and argue that it is acceptable since the loss of a secret key often means a total break of the system.

5.5 Zheng's Scheme

The first signcryption scheme is due to Zheng's great idea to merge a variant of the ElGamal signature scheme with a non-interactive Diffie-Hellman key exchange. The signature scheme is described in Figure 5.1, Zheng realized that in such a signature, the first part of the verification procedure is to retrieve a group element g^r (without knowing r) from

Public parameters:

- Cyclic group G of order q generated by g
- Symmetric encryption scheme E with keyspace $\mathcal{K}$
- Encoding function H mapping bitstrings to $\mathbb{Z}_q$
- Key derivation function $\mathsf{KDF} : \{0,1\}^* \to \mathcal{K}$

Alice's keys: $sk_a \xleftarrow{r} \mathbb{Z}_q; pk_a \leftarrow g^{sk_a}$
Bob's keys: $sk_b \xleftarrow{r} \mathbb{Z}_q; pk_b \leftarrow g^{sk_b}$

Zheng Signcrypt(sk_a, pk_a, pk_b, m):

1: $r \xleftarrow{r} \mathbb{Z}_q$
2: $K \leftarrow \mathsf{KDF}(pk_b^r)$
3: $c \leftarrow \mathsf{H}(m||pk_a||pk_b||pk_b^r)$
4: $z \leftarrow r \cdot (sk_a + c)^{-1}$
5: $\mathcal{E} \leftarrow \mathsf{E}(K, m)$
6: **return** $c, z, \mathcal{E}$

Zheng Unsigncrypt(c, z, sk_b, pk_a, pk_b):

1: $\omega \leftarrow (pk_a \circ g^c)^z \ (= g^r)$
2: $K \leftarrow \mathsf{KDF}(\omega^{sk_b})$
3: $m \leftarrow E^{-1}(K, \mathcal{E})$
4: $verif \leftarrow c \stackrel{?}{=} \mathsf{H}(m||pk_a||pk_b||\omega^{sk_b})$
5: **return** m **if** $verif$ **else** $\perp$

Figure 5.2: Original Zheng's signcryption scheme

public values. The interesting point is that communicating such a value is exactly the first part of a Diffie-Hellmann key exchange (Alice chooses a and sends g^a). The value (c, z) are actually not only a signature, but also a Alice's side of a key exchange. Thus, if Bob previously published a public key of the form $pk_b = g^{sk_b}$, Alice can compute a one-time shared secret key $K = pk^r$ that Bob will reconstruct by computing $K = \omega^{sk_b}$. This key can be used to symmetrically encrypt a message that will be append to the signature to get a scheme performing signature and encryption of a message at the same time and in which the performance gain comes from sharing the nonce r between the signature and the key exchange. The signcryption scheme of Zheng is depicted in Figure 5.2.

5.5.1 Schnorr Variant

The work of Zheng led to a plethora of variants (see [45], Part II). Among them, a very natural extension of this idea of reusing nonce was to apply it to Schnorr signature. This was proposed by Malone-Lee in [86]. Indeed, similarly to the variant of ElGamal signature described above, in Schnorr signature, the signer also picks a random r and the verifier also retrieves g^r. We give Malone-Lee's scheme in Figure 5.3. The constructions follows naturally from the one of Zheng, exposed in the previous section. The reason why this scheme is very interesting to us is that the lattice-based signatures studied in this thesis are themselves based on Schnorr signatures. It was thus easier to derive a post-quantum signcryption scheme from the proposal of Malone-Lee than from Zheng's version. One reason is that Zheng's sigcryption scheme has an inversion modulo q which does not translate to anything concrete in the lattice setting.

Public parameters:

- Cyclic group G of order q generated by g
- Symmetric encryption scheme E with keyspace $\mathcal{K}$
- Encoding function H mapping bitstrings to Z_q
- Key derivation function $\mathsf{KDF} : \{0,1\}^* \to \mathcal{K} \times \{0,1\}^\ell$

Alice's keys: $sk_a \xleftarrow{r} \mathbb{Z}_q; pk_a \leftarrow g^{sk_a}$
Bob's keys: $sk_b \xleftarrow{r} \mathbb{Z}_q; pk_b \leftarrow g^{sk_b}$

Malone-Lee Signcrypt(sk_a, pk_a, pk_b, m):

1: $y \xleftarrow{r} \mathbb{Z}_q$
2: $K_1, K_2 \leftarrow \mathsf{KDF}(pk_b^y)$
3: $c \leftarrow \mathsf{H}(m||pk_a||pk_b||g^y||K_2)$
4: $z \leftarrow y - sk_a \cdot c$
5: $\mathcal{E} \leftarrow \mathsf{E}(K_1, m)$
6: **return** $c, z, \mathcal{E}$

Malone-Lee Unsigncrypt(c, z, sk_b, pk_a, pk_b):

1: $\omega \leftarrow g^z \circ pk_a^c \ (= g^y)$
2: $K_1, K_2 \leftarrow \mathsf{KDF}(\omega^{sk_b})$
3: $m \leftarrow E^{-1}(K_1, \mathcal{E})$
4: $verif \leftarrow c \stackrel{?}{=} \mathsf{H}(m||pk_a||pk_b||\omega||K_2)$
5: **return** m **if** $verif$ **else** $\perp$

Figure 5.3: Malone-Lee's signcryption scheme. Based on Schnorr signature.

5.6 Reconciliation Mechanism

A common issue in learning with errors key exchanges [12, 27, 28, 47, 100] is that both parties end up with two values that are close to each other but not exactly the same. It is due to the fact that, as in the encryption scheme, it is often made of ElGamal-like cryptography but with noisy elements. For example in the RLWE version, Alice eventually computes $ass' + e's$ while Bob has $ass' + es'$. Obviously, the key exchange cannot be considered successful if each party has a different value. The solution is to use a reconciliation mechanism deriving a common value from noisy data (a.k.a fuzzy extractor [48]).

Clearly, any reconciliation technique has an error tolerance threshold over which agreement cannot be reached. To increase the threshold, a possibility is to use *multiple* values to agree on a common bit. The motivation is that polynomials used in RWLE-based are often of size 512 or 1024 to ensure the security of the underlying lattice problem while symmetric secrets of bit size 256 appear to be enough, even in a post-quantum world. Hence we should use mappings from $\mathbb{Z}_q^n$ to $\{0,1\}^{256}$ with $n \in \{512, 1024\}$. Of course mappings for higher n or larger symmetric keys can be used but in practice, those parameters are good enough. For the key exchange version of our construction, we borrow the notations from NEWHOPE [12]. In their paper, they show how to agree on a n bit key from either a polynomial of degree $2n$ or $4n$. The description of their whole reconciliation mechanism is quite tedious and takes a lot of space. Hence we redirect the interested reader to their paper for a full explanation and analysis. By borrowing their notations, we mean that we will use two algorithms $\texttt{HelpRec}(x)$ and $\texttt{Rec}(x', \mathbf{r})$ (as defined below) but that the scheme is unaffected by how those functions work under the hood, they could implement any reconciliation mechanism.

- $\texttt{HelpRec}(x)$ taking as input a ring element and outputting a reconciliation vector $\mathbf{r}$

- $\texttt{Rec}(x', \mathbf{r})$ taking as input a ring element and a reconciliation vector and outputting a symmetric key K

If x and x' are close to each other (the distance between their coefficients is small), the output of $\texttt{Rec}(x, \mathbf{r})$ and $\texttt{Rec}(x', \mathbf{r})$ are the same.

5.7 SETLA: Signature and Encryption from Lattices (CANS 2018)

Hereunder, we describe both versions of our scheme. The discussion in Section 5.9 will only be made for the first version for the sake of brevity but the analysis is basically the same. In the following, when we talk about lattice signatures, we mean lattice-based signatures obtained from the Fiat-Shamir transformation.

5.7.1 SETLA-KEX Signcryption

First we describe how to integrate encryption into a lattice signature, following the steps of the ElGamal modification of Zheng. From a high-level point of view, the idea of the original signcryption scheme is to sign a message with an ElGamal signature and to perform a non-interactive Diffie-Hellman ephemeral key exchange (KEX) at the same time reusing the "commit" value of the signature. The gain in efficiency comes from the fact that the same operation is used in both primitives. Subsequently, the message is symmetrically encrypted with the key derived from the exchange and forwarded to the receiver. While the first scheme of Zheng was not directly translatable in a lattice version, the scheme Malone-Lee is a good candidate. Indeed, even though its primary advantage over Zheng in pre-quantum cryptography was to enable non-interactive non-repudiation, namely that Bob alone can create a valid signature from a signcryptext, the second difference is that it is based on Schnorr signature. A explained above, the lattice-based signatures schemes coming from identification schemes through Fiat-Shamir transform being Schnorr-like [10, 15, 49, 81], this is where post-quantum can meet signcryption. We use a ring version of the signature proposed by Bai and Galbraith as a base to construct the scheme but it can be generalized to most signatures derived from the original work of Lyubashevsky as long as the parameters offer at the same time security and correctness for the key reconciliation. We actually also have a construction based on GLP working out of the box with the original parameters which is omitted since the scheme is considered widely deprecated now.

Algorithm 34 SETLA Key generation

Input: Public parameter $a_1, a_2 \in \mathcal{R}_q$
Output: Key pair $pk = (t_1, t_2), sk = (s, e_1, e_2)$

1: $s, e_1, e_2 \xleftarrow{r} \mathcal{R}_{q,[1]}$
2: $t_1 \leftarrow a_1 \cdot s + e_1$, $t_2 \leftarrow a_2 \cdot s + e_2$
3: **return** $pk = (t_1, t_2), sk = (s, e_1, e_2)$

Key generation (Algorithm 34).

The key generation is simple and straightforward for a scheme using ideal lattices cryptography. It uses some public parameters a_1, a_2 shared among all users and output two RLWE samples $pk = (t_1, t_2)$ together with a secret polynomial s. The error and secret distributions are the same and output a polynomial with uniform coefficients in $\{-1, 0, 1\}$. The choice of such a distribution is suboptimal in terms of security since it has low variance and its special structure may enable specialized attacks [117] but has been made for reasons that will come clear later. Note that in the context of signcryption, both Alice and Bob will run the key generation procedure to retrieve their keys since two key pairs are used in the full signcrypt/unsigncrypt procedure. In the following, we use subscripts, e.g. $pk_a = (t_{a,1}, t_{a,2})$, to differentiate them.

Algorithm 35 SETLA-KEX Signcrypt

Input: Public parameters a_1, a_2, Bob's public key pk_b, Alice's keys $(s_a, e_{a,1}, e_{a,2}, pk_a)$, a message m, random oracle $H : * \to \{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = \omega\}$, symmetric encryption algorithm E

Output: a signcryptext of m: $C = (z, c, \mathcal{E}, r)$

do
  $y, y' \xleftarrow{r} \mathcal{R}_{q,[B]}$
  $v \leftarrow t_{b,1} \cdot y + y' = a_1 \cdot s_b \cdot y + e_{b,1} \cdot y + y'$
  $\mathbf{r} \leftarrow \mathtt{HelpRec}(v)$
  $K \leftarrow \mathtt{Rec}(v, \mathbf{r})$
  $c \leftarrow H(\lfloor a_1 \cdot y \rceil_d, \lfloor a_2 \cdot y \rceil_d, m, K, pk_a, pk_b)$
  $z \leftarrow s_a \cdot c + y$
  $w_1 \leftarrow a_1 \cdot y - e_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot y - e_{a,2} \cdot c$
while not(z **in** $\mathcal{R}_{q,[B-\omega]}$ **and** $\lfloor a_1 \cdot y \rceil_d = \lfloor w_1 \rceil_d$ **and** $\lfloor a_2 \cdot y \rceil_d = \lfloor w_2 \rceil_d$ **)**
$\mathcal{E} \leftarrow \mathsf{E}(K, m)$
return $z, c, \mathcal{E}, r$

SETLA-KEX Signcrypt (Algorithm 35).

The signcrypt procedure contains three interleaved parts: signature, key exchange and encryption. The signature follows the structure of [2, 16] as a Fiat-Shamir signature from a sigma protocol. First, a commitment consisting of two rounded polynomials $\lfloor a_1 \cdot y \rceil_d, \lfloor a_2 \cdot y \rceil_d$ depending on a masking value y is computed. Then, an unpredictable challenge c is retrieved by simulating a verifier with a random oracle H taking inputs depending on the commitment. Finally, the response consists of a polynomial of the form $z = s \cdot c + y$. Note that for reasons related specifically to signcryption schemes, the random oracle should take as input a symmetric key K and both public identities. If the key were not included in the input, the adversary playing a signcryption specific CCA2 game would easily be able to distinguish between two messages m_0, m_1 by computing both $H(., m_i, ., .)$ and verifying the equality with c. Having the public identities in the hash is a common practice in signcryption schemes to prove security in advanced models [45].
The key exchange part is performed by deriving a secret value K from a noisy version of $a_1 \cdot s_b \cdot y$. Alice cannot find the exact value since it would mean she knows Bob's secret key but she can find an approximate value from Bob's public key by computing

$t_{b,1} \cdot y = a_1 \cdot s_b \cdot y + e'_{b,1} \cdot y \approx a_1 \cdot s_b \cdot y$. This is exactly the technique employed in lattice-based key exchanges such as NEWHOPE. The efficiency gain comes from the fact that Bob will later be able to retrieve an approximation version of $a_1 \cdot y$ *without* sending him any other ring element than the polynomials computed in the signature (z, c). As in [12,28], Alice gets a symmetric key by applying a reconciliation procedure on the noisy shared value. The last part is straightforward, now that a key is available, a symmetric cipher E is used to encrypt the data.
Finally, Alice outputs the signature (z, c), the symmetric ciphertext $\mathcal{E}$ and a small reconciliation vector $\mathbf{r}$. It means that the message was at the same time encrypted and authenticated in an asymmetric manner with only the overhead of sending a symmetric ciphertext (obviously we need to send something at least as long as the message for encryption) and a small reconciliation vector on the top of the signature.

Algorithm 36 SETLA-KEX Unsigncrypt

Input: Public parameters a_1, a_2, Bob's keys (s_b, pk_b), Alice's public key pk_a, a signcryptext $C = (z, c, \mathcal{E}, \mathbf{r})$, random oracle $H : * \rightarrow \{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = \omega\}$, symmetric encryption algorithm E
Output: A message m or failure symbol $\perp$

1: $w_1 \leftarrow a_1 \cdot z - t_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot z - t_{a,2} \cdot c$
2: $K \leftarrow \mathtt{Rec}(w_1 \cdot s_b, \mathbf{r})$
3: $m \leftarrow \mathsf{E}_K^{-1}(\mathcal{E})$
4: **return** m **if** $c = H(\lfloor w_1 \rceil_d, \lfloor w_2 \rceil_d, m, K, pk_a, pk_b)$ and $z \in \mathcal{R}_{q,[B-\omega]}$ **else** $\perp$

SETLA-KEX Unsigncrypt (Algorithm 36).

The goal of this algorithm is to allow Bob to find the secret key to decrypt the symmetric cipher and at the same, to provide authentication of the message through a signature.
First, Bob will recover the commitment part of the signature by rounding the values $w_1 \leftarrow a_1 \cdot z - t_{a,1} \cdot c$ and $w_2 \leftarrow a_2 \cdot z - t_{a,2} \cdot c$. Without rounding, c would have been different since Alice queried the random oracle with $\lfloor a_1 \cdot y \rceil_d$ and $\lfloor a_2 \cdot y \rceil_d$. The difference with the original signature scheme is that Bob must now find the key K and the message in order to verify the hash value. To recover it, he shall use the reconciliation vector $\mathbf{r}$ with an approximate version of $a_1 \cdot s_b \cdot y$. Such a value can be found by computing the product $w_1 \cdot s_b = a_1 \cdot s_b \cdot y + e_{a,1} \cdot s_b \cdot c \approx a_1 \cdot s_b \cdot y$. Once the message is decrypted, Bob verifies the signature by checking the size of z and the hash value. He outputs the message if everything is correct and a failure symbol otherwise.

SETLA-KEX Signature Extraction (Algorithm 37).

An interesting feature of Malone-Lee's signcryption scheme is that the receiver Bob can himself create a fully valid publicly verifiable signature under Alice's secret key on the message he unsigncrypted. Our scheme also inherits this capability. Even if we chose to start from this scheme for its similarity with Schnorr signature (and thus, lattice-based signatures), this really helpful feature carries to our construction. The corresponding verification algorithm is very similar to Algorithm 23 and basically the same as the verification algorithm in [16], the verifier computes w_1 and w_2 as in Algorithm 37 and accepts if $c = H(\lfloor w_1 \rceil_d, \lfloor w_2 \rceil_d, m, K, pk_a, pk_b)$ and $z \in \mathcal{R}_{q,[B-\omega]}$.

Algorithm 37 SETLA-KEX SignExtract

Input: Public parameters a_1, a_2, Bob's keys (s_b, pk_b), Alice's public key pk_a, a signcryptext $C = (z, c, \mathcal{E}, \mathbf{r})$, random oracle $H : * \to \{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = \omega\}$, symmetric encryption algorithm E

Output: A message m together with its signature $\sigma(m)$ or a failure symbol

1: $w_1 \leftarrow a_1 \cdot z - t_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot z - t_{a,2} \cdot c$
2: $K \leftarrow \mathtt{Rec}(w_1 \cdot s_b, \mathbf{r})$
3: $m \leftarrow \mathsf{E}_K^{-1}(\mathcal{E})$
4: $b \leftarrow c = H(\lfloor w_1 \rceil_d, \lfloor w_2 \rceil_d, m, K, pk_a, pk_b)$ **and** $z \in \mathcal{R}_{q,[B-\omega]}$
5: **return** $m, \sigma(m) = (K, z, c)$ **if** $b = 1$ **else** $\perp$

5.7.2 SETLA-KEM Signcryption

Now, we describe the second version of the scheme based on key encapsulation instead of direct key exchange. The approach is similar to the one first used in NEWHOPE-SIMPLE [11] and now in the NIST submission or KYBER. The high-level perspective is now to perform a noisy ElGamal encryption of a chosen key during signature instead of noisy Diffie-Hellman. While in NEWHOPE-SIMPLE the goal of the new approach is to make the protocol simpler by getting rid of the reconciliation mechanism but not really to enhance the scheme, here, using an encryption based method leads to better performances in terms of speed and can enable parallelism, at the cost of a significantly larger signcryptext.

Algorithm 38 SETLA-KEM Signcrypt

Input: Public parameters a_1, a_2, Bob's public key pk_b, Alice's key $(s_a, e_{a,1}, e_{a,2}, pk_a)$, a message m, random oracle $H : * \to \{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = \omega\}$, symmetric encryption algorithm E

Output: a signcryptext of m: $C = (z, c, x, \mathcal{E})$

1: $K \xleftarrow{r} \{0,1\}^{256}$
2: **do**
3: $\quad y \xleftarrow{r} \mathcal{R}_{q,[B]}$
4: $\quad c \leftarrow H(\lfloor a_1 \cdot y \rceil_d, \lfloor a_2 \cdot y \rceil_d, m, K, pk_a, pk_b)$
5: $\quad z \leftarrow s_a \cdot c + y$
6: $\quad w_1 \leftarrow a_1 \cdot y - e_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot y - e_{a,2} \cdot c$
7: **while not(** z **in** $\mathcal{R}_{q,[B-\omega]}$ **and** $\lfloor a_1 \cdot y \rceil_d = \lfloor w_1 \rceil_d$ **and** $\lfloor a_2 \cdot y \rceil_d = \lfloor w_2 \rceil_d$ **)**
8: $y' \xleftarrow{r} \mathcal{R}_{q,[B]}$
9: $x \leftarrow t_{b,1} \cdot y + y' + \mathtt{Encode}(K)$
10: $\mathcal{E} \leftarrow \mathsf{E}(k, m)$
11: **return** $z, c, x, \mathcal{E}$

SETLA-KEM Signcrypt (Algorithm 38).

In the same way as before, one can find three phases: signature, key encapsulation and symmetric encryption. The signature is now more isolated and almost exactly the same as in [16], the small difference is that the random oracle (as in the KEX version) takes as input the message, the symmetric decryption key and the public identities.
The key encapsulation part is a RLWE encryption (similar to NEWHOPE) of a randomly

sampled key K. Like ElGamal (Figure 2.2), such an encryption consists of two values $c_1 = a \cdot y_1 + y_2$ and $c_2 = pk_b \cdot y_1 + y_3 + \mathsf{Encode}(m)$ ($\hat{u}$ and v' in Figure 3.4). Basically, c_2 is the message masked with a ring element depending on the public-key looking random under the decisional-RLWE assumption and c_1 is a value allowing the owner of s_b to remove the mask without conveying any (computable) information on y_1 under the search-RLWE assumption. Here we gain efficiency by having the value $\lfloor a_1 \cdot y \rceil_d$ acting at the same time as the commitment of the signature and the c_1 part of the encryption scheme. The c_2 part is given by x.

Globally, the KEM version is adding a lot of overhead on the size of the signcryptext which is problematic since this is where we are looking for efficiency. Nevertheless, we see two advantages of using encryption instead of key exchange. First, the scheme is faster because it has less computation in the rejection sampling loop (which can run several times depending on the parameters) and we can now parallelize the symmetric encryption algorithm. Indeed, in the KEX version, the key depends on y and was not known until the end of the rejection sampling procedure, hence, everything had to be sequential and a multiplication with $t_{b,1}$ had to be done at each iteration. Now, the symmetric encryption can start at the same time as the rejection sampling. It is fair to say that in general symmetric operations are lightweight in comparison to polynomial multiplication. Nevertheless, if a really large message has to be encrypted, say such that $E_K(m)$ takes as long as the **do**...**while** loop, the saving becomes non-negligible. Obviously, this argument only makes sense if the rejection sampling procedure itself is not affected by the size of the message. One solution would be to pre-hash the message before the loop and only inject this hash in the random oracle. Actually this issue is not specific to signcryption, all the signature schemes using rejection sampling would be badly affected by a really long message if the hash function cannot restart from its previous sate. Hence, in this case, hashing the message once before would save some computation. This small modification could be done in the KEX version as well as in existing Fiat-Shamir lattice-based signatures.

Second, depending on the parameters, if the correctness is an issue, having the key encoded as a polynomial with coefficients in $\{0, \frac{q-1}{2}\}$ is optimal for the reconciliation since they are at"maximum distance" in $\mathbb{Z}_q$. Also, because the symmetric key needed being often smaller than the encoding polynomial, having control over the value eases the process of embedding an error-correcting code in the extra space. Even though in the current state of affairs and with the parameters proposed in Section 5.9 the KEM version would not outperform neither the KEX version nor the naive concatenation of efficient schemes, we think the construction may be of interest in some contexts.

Algorithm 39 SETLA-KEM Unsigncrypt

Input: Public parameter a_1, a_2, Bob's key (s_b, s'_b, pk_b), Alice's public key pk_a, a signcryptext $C = (z, c, x, \mathcal{E})$, random oracle $H : * \to \{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = \omega\}$, symmetric encryption algorithm E

Output: A message m or failure symbol $\perp$

1: $w_1 \leftarrow a_1 \cdot z - t_{a,1} \cdot c$
2: $w_2 \leftarrow a_2 \cdot z - t_{a,2} \cdot c$
3: $K \leftarrow \mathsf{Decode}(x - w_1 \cdot s_b)$
4: $m \leftarrow \mathsf{E}^{-1}(\mathcal{E})$
5: **return** m **if** $c = H(v, m, K, pk_a, pk_b)$ and $z \in \mathcal{R}_{q,[k-\omega]}$ **else** $\perp$

SETLA-KEM Unsigncrypt (Algorithm 39).

The unsigncrypt algorithm follows in the obvious manner. Bob retrieves the c_1 part of the RLWE encryption from the signature and run the decryption algorithm to find the key. Then, he decrypts the symmetric ciphertext and verifies the signature.

5.8 Security Arguments

The security aspects of interest for signcryption are unforgeability and privacy. The construction combining both a signature scheme using the Fiat-Shamir heuristic and a public key encryption scheme, we argue the security by using the forking lemma [105] and a standard hybrid argument. This does not provide a formal argument of security in a signcryption specific security model since it does not consider the primitive as an encryption *and* a signature but rather successively as an encryption *or* a signature. We do not claim that this is a sufficient analysis, nevertheless, having both unforgeability and privacy of the two underlying schemes is a good pointer toward the fact the design is sound. Providing a formal argument in advanced signcryption models is a tedious task (see [14]) and we do not attempt to do so here.

5.8.1 Unforgeability

The underlying signature of the signcryption scheme is the ring variant of the Bai-Galbraith signature which is itself a derivative of the original proposal of Lyubashevsky [81]. The full security argument can be found in [15] but the idea is to use the forking lemma to get two different signatures for the same commitment that would allow us to solve a special SIS instance. We use the adversary to get two forgeries z, c and z', c' for different random oracles but the same random tape (hence the same y). We have (providing the argument for only one RLWE sample instead of two as in the signature for the sake of simplicity) $\lfloor a \cdot z - t_a \cdot c \rceil_d = \lfloor a \cdot z' - t_a \cdot c' \rceil_d = \lfloor a \cdot y \rceil_d$. This means that for some small e, $a \cdot z - t_a \cdot c = a \cdot z' - t_a \cdot c' + e$ and thus, with $t_a = a \cdot s_a + e_a$, $a \cdot (z - z' - s_a \cdot c + s_a \cdot c') + (e_a \cdot (c' - c) + e) = 0$. As pointed in [15] section 4.2, (if $z - z' - s_a \cdot c + s_a \cdot c'$ and $e_a \cdot (c' - c) + e$ are non-zero) we have found a solution to the SIS instance. This argument still holds for the signcryption scheme.

5.8.2 Confidentiality

We argue the confidentiality of the scheme with a sequence of games showing semantic security under the Decisional Compact Knapsack assumption in the random oracle model. We model the adversary as a tuple of two algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the first choosing messages for the game according to the public keys and the second trying to guess which one was signcrypted. The encryption scheme E is seen as an ideal primitive. The sequence of games for the KEX version can be found in Figure 5.4. Games for the KEM version are really similar.

Game 0: Game 0 is the usual CPA game against SETLA, the adversary chooses two messages m_0, m_1 and tries to guess which one was signcrypted.

Game 0:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk_a, pk_b)$
$b \xleftarrow{r} \{0,1\}$
$y, y' \xleftarrow{r} \mathcal{R}_{q,[B]}$
$v \leftarrow t_{b,1} \cdot y + y'$
$\mathbf{r} \leftarrow \texttt{HelpRec}(v)$
$K \leftarrow \texttt{Rec}(v, \mathbf{r})$
$h_1 \leftarrow \lfloor a_1 \cdot y \rceil_d, h_2 \leftarrow \lfloor a_2 \cdot y \rceil_d$
$c \leftarrow H(h_1, h_2, m, K, pk_a, pk_b)$
$z \leftarrow s_a \cdot c + y$
$w_1 \leftarrow a_1 \cdot y - e_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot y - e_{a,2} \cdot c$
if $h_1 \neq \lfloor w_1 \rceil_d$ **or** $h_1 \neq \lfloor w_2 \rceil_d$, **goto** 3
if z **not in** $\mathcal{R}_{q,[B-\omega]}$, **goto** 3
$\mathcal{E} \leftarrow \mathsf{E}(k, m)$
$\hat{b} \leftarrow \mathcal{A}_2(z, c, \mathcal{E}, \mathbf{r})$
return $\hat{b}$

Game 1:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk_a, pk_b)$
$b \xleftarrow{r} \{0,1\}$
$y, y' \xleftarrow{r} \mathcal{R}_{q,[B]}$
$v \leftarrow t_{b,1} \cdot y + y'$
$\mathbf{r} \leftarrow \texttt{HelpRec}(v)$
$K \leftarrow \texttt{Rec}(v, \mathbf{r})$
$h_1 \leftarrow \lfloor a_1 \cdot y \rceil_d, h_2 \leftarrow \lfloor a_2 \cdot y \rceil_d$
$c \leftarrow H(h_1, h_2, m, K, pk_a, pk_b)$
$z \xleftarrow{r} \mathcal{R}_{q,[B-\omega]}$
$w_1 \leftarrow a_1 \cdot y - e_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot y - e_{a,2} \cdot c$
if $h_1 \neq \lfloor w_1 \rceil_d$ **or** $h_1 \neq \lfloor w_2 \rceil_d$, **goto** 3
with probability P, **goto** 3
$\mathcal{E} \leftarrow \mathsf{E}(k, m)$
$\hat{b} \leftarrow \mathcal{A}_2(z, c, \mathcal{E}, \mathbf{r})$
return $\hat{b}$

Game 2:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk_a, pk_b)$
$b \xleftarrow{r} \{0,1\}$
$y, y' \xleftarrow{r} \mathcal{R}_{q,[B]}$
$a' \xleftarrow{r} \mathcal{R}_q$
$v \leftarrow a' \cdot y + y'$
$\mathbf{r} \leftarrow \texttt{HelpRec}(v)$
$K \leftarrow \texttt{Rec}(v, \mathbf{r})$
$h_1 \leftarrow \lfloor a_1 \cdot y \rceil_d, h_2 \leftarrow \lfloor a_2 \cdot y \rceil_d$
$c \leftarrow H(h_1, h_2, m, K, pk_a, pk_b)$
$z \xleftarrow{r} \mathcal{R}_{q,[B-\omega]}$
$w_1 \leftarrow a_1 \cdot y - e_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot y - e_{a,2} \cdot c$
if $h_1 \neq \lfloor w_1 \rceil_d$ **or** $h_1 \neq \lfloor w_2 \rceil_d$, **goto** 3
with probability P, **goto** 3
$\mathcal{E} \leftarrow \mathsf{E}(k, m)$
$\hat{b} \leftarrow \mathcal{A}_2(z, c, \mathcal{E}, \mathbf{r})$
return $\hat{b}$

Game 3:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk_a, pk_b)$
$b \xleftarrow{r} \{0,1\}$
$y, y' \xleftarrow{r} \mathcal{R}_{q,[B]}$
$v \xleftarrow{r} \mathcal{R}_q$
$\mathbf{r} \leftarrow \texttt{HelpRec}(v)$
$K \leftarrow \texttt{Rec}(v, \mathbf{r})$
$h_1 \leftarrow \lfloor a_1 \cdot y \rceil_d, h_2 \leftarrow \lfloor a_2 \cdot y \rceil_d$
$c \leftarrow H(h_1, h_2, m, K, pk_a, pk_b)$
$z \xleftarrow{r} \mathcal{R}_{q,[B-\omega]}$
$w_1 \leftarrow a_1 \cdot y - e_{a,1} \cdot c$, $w_2 \leftarrow a_2 \cdot y - e_{a,2} \cdot c$
if $h_1 \neq \lfloor w_1 \rceil_d$ **or** $h_1 \neq \lfloor w_2 \rceil_d$, **goto** 3
with probability P, **goto** 3
$\mathcal{E} \leftarrow \mathsf{E}(k, m)$
$\hat{b} \leftarrow \mathcal{A}_2(z, c, \mathcal{E}, \mathbf{r})$
return $\hat{b}$

Figure 5.4: Sequence of games for the KEX version

Game 1: By virtue of the rejection sampling performed during signcryption, the output distribution of z should be exactly the same as a uniform over $\mathcal{R}_{q,[k-\omega]}$. Hence, we can replace z by random elements over this range without modifying the view of the adversary.

Game 2: Using the RLWE assumption, we can replace the public key of Bob by a random element in $\mathcal{R}_q$ without being detected by the polynomial time adversary.

Game 3: In game 3, we use the same argument again to replace v by a uniformly random value (and hence K is uniform as well by design of Rec).

In conclusion, using the fact that both $H(.)$ and $E(.)$ are modeled as ideal primitives

and that H takes one random unknown to the adversary value (K) uncorrelated to the message, they do not reveal anything about their inputs. Hence, the values given to $\mathcal{A}_2$ looks all random and independent from the messages. Thus, the adversary cannot guess which one was signcrypted.

ROM vs QROM

It is known that the forking lemma cannot be used if the adversary has quantum access to the random oracle. This issue has been recently discussed a lot in the literature on lattice-based signatures and some schemes took it into consideration [10] while others ignored it to focus on performances [50]. Since our goal is to improve practicability, we decided to stick to the classical ROM. Having a classical reduction is essential to claim provable security but the implications of the QROM issue in practice are not clear enough to require QROM security for all the schemes. We redirect the interested reader to [26,72] for more details.

5.9 Analysis and Parameters

5.9.1 Parameters Selection

To select the parameters, we followed the methodology described in [10]. It allows the scheme to reduce to worst-case problems on ideal lattices. Be careful that it does not mean that the parameters are chosen such that the problem we reduce to is hard (since the proofs are non-tight) but merely that the reduction works. This is a common practice and as pointed in [72], it is reasonable to assume that it does not create any security issue. Our parameters can be found in Table 5.1. The dimension n has been set to 1024 because it seems to be the minimal lattice dimension such that RLWE is hard with such a small error distribution. The value m represents the number of rows of the LWE instance written in matrix form. Here it means that we work with two polynomials (which are explicit in the construction) since $m = 2n$. The entropy of the output of the random oracle is given by $\kappa = \log_2\left(2^\omega \binom{n}{\omega}\right)$, that is to say the logarithm of the cardinality of the set $\{v \mid v \in \mathcal{R}_{q,[1]}, \|v\|_1 = \omega\}$. The modulus $q = 2^{25} - 2^{12} + 1$ is a prime such that $q \equiv 1 \mod 2n$ .The parameters d and B are chosen such that the acceptance probability of the signature is not too low ($\delta \approx 0.56$) in order to keep the runtime reasonable and $q^{m-n} \geq \frac{2^{(d+1)m+\kappa}}{(2B)^n}$ as required in the security proof of [15].
To assess the security of the scheme, we used the *LWE-Estimator* tool of Albrecht and al. [5]. We ran the estimator with the following command:

```
n = 1024; q = 33550337;
stddev = sqrt(2/3); alpha = alphaf(sigmaf(stddev), q)
_ = estimate_lwe(n, alpha, q,
 secret_distribution=(-1,1),reduction_cost_model=BKZ.sieve)
```

It estimates a bit security of 131 against the most efficient attack. The estimation of the hardness of directly forging the signature without recovering the private key has been made in the same way as in [43]. It gave overwhelming results, which is not a surprise since the parameters are a harder version of the most secure parameters set of [63].

n	m	ω	d	B	q	κ
1024	2048	16	15	2^{15}	$33550337 \approx 2^{25}$	131

Table 5.1: Parameters targeting 128 bits of classical security. We only claim *classical* security of 128 bits because we use a 128-bit hash (similar to [16])

5.9.2 Failure Probability

The main bottleneck of the signcryption scheme is the correctness regarding decryption. Indeed, as in a lot of RLWE-based protocols, the two parties end up with two ring elements close to each other but not exactly the same. In our case, the difference between the value of Alice and the value of Bob is $\Delta_{ab} = e_{b,1} \cdot y - e_{a,1} \cdot c \cdot s_b + y'$. While in those schemes the parameters are chosen in order to get correctness with overwhelming probability, we face here a strong constraint which is that the parameters should also be compatible with the signature scheme. In their case, the y is coming from the error distribution and hence is very small. In our case, it is the masking polynomial for the signature $s \cdot c + y$ which should be significantly larger. Obviously, one strategy to reduce the norm of Δ_{ab} is to reduce B. This would give better results for correctness but unfortunately decrease the speed of the scheme since the rejection sampling loop would have to run longer to find a small enough z. This is the reason why we decided to use such a small distribution for the secret and the errors. Of course, it is possible to work with slightly larger distributions in a more specific context in which correctness matters less.

We now provide an analysis of the failure probability for the KEX-version. Using the reconciliation method of NewHope, the KEX-Unsigncrypt algorithm recovers the correct key if $\|\Delta_{ab}\|_\infty < \left\lfloor \frac{3q}{8} \right\rfloor$ (actually the requirement is that the ℓ_1 norm of packs of 4 coefficients should be smaller than $\left\lfloor \frac{3q}{4} \right\rfloor - 2$). In the following, we write $(p)_i$, to denote the i-th coefficient of a polynomial p.
We shall bound the magnitude of one coefficient $(\Delta'_{ab})_i = (e'_{b,1} \cdot y)_i$. Since the polynomial product is computed modulo $\langle X^n + 1 \rangle$ and all distributions are symmetric, one such coefficient is the result of a sum of n products between a coefficient of a polynomial in $\mathcal{R}_{q,[1]}$ and a polynomial in $\mathcal{R}_{q,[B]}$.

Let $S \sim \mathcal{U}(\{-1, 0, 1\})$ and $Y \sim \mathcal{U}([-B, B])$ be random variables, we denote their product SY. Each coefficient of Δ'_{ab} is the sum of n samples from SY, hence $(\Delta'_{ab})_i \sim \sum_{i=1}^{n} (SY)_i$. Fortunately, computing the exact distribution SY is easy:

$$Pr[SY = 0] = \frac{2B + 3}{6B + 3}$$

$$Pr[SY = z \mid z \in [-B, B] \backslash \{0\}] = \frac{2}{6B + 3}$$

Since the value of B is reasonable, to find the distribution of Δ'_{ab}, one could hope to compute $\log(n)$ time the convolution of the distribution with itself. Unfortunately, this approach failed to give accurate results because of numerical stability issues. Instead, as in [12], we use the Chernoff-Cramer inequality to bound the sum of the random variables.

Chernoff-Cramer inequality Let χ be a distribution over $\mathbb{R}$ and let $X_1, \ldots, X_n$ be i.i.d. symmetric random variables of law χ. Then, for any t such that $M_\chi(t) = \mathbb{E}[e^{tX}] < \infty$ it holds that

$$Pr\left[\left|\sum_{i=1}^{n} x_i\right| > \alpha\right] < 2e^{-\alpha t + n\log(M_\chi(t))}.$$

Using the above inequality with

$$M_{SY}(t) = \frac{2B+3}{6B+3} + \frac{2}{6B+3} \cdot \left(\frac{e^{t(B+1)}-1}{e^t-1} + \frac{e^{-t(B+1)}-1}{e^{-t}-1} - 2\right)$$

and setting $\alpha = \left\lfloor \frac{q}{4} \right\rfloor - B - n\omega$, $t \approx 2.5 \cdot 10^{-5}$, $n = 1024$ and $k = 2^{15}$ (our parameters from the previous section), we find that $Pr\left[(\Delta'_{ab})_i > \left\lfloor \frac{3q}{16} \right\rfloor\right] \approx 2^{-115}$. By virtue of the union bound on the 1024 coefficients, we get that the failure probability is at most $\approx 2^{-105}$.

5.9.3 Performances

Even if the construction of the signcryption scheme is conceptually interesting on its own, its usage only makes sense if we gain something over the trivial solution of concatenating an encryption/key exchange and a signature scheme. In Table 5.2, we compare the performances regarding bandwidth between SETLA and a selection of schemes of the same kind. Since lattice-based schemes are already doing great in terms of speed, especially when they can take advantage of SIMD (Single Instruction Multiple Data) instructions, reducing bandwidth will a major factor for adoption in the future. We decided to compare SETLA to the pairs given in the table for the following reasons:

- DILITHIUM + KYBER: They were very recently designed and are part of the same family of algorithms.
- QTESLA + NEWHOPE: They are the two up-to-date RLWE based schemes and are both candidates for future standardization.
- TESLA♯ + KYBER: This seems to be the most efficient pair regarding compactness out of the reasonably secure Fiat-Shamir/key exchange schemes in the literature.

For the record, we also indicates the performances of the GLP version of signcryption that is using the original parameters of the signature [63]. It obviously gives goods results since we get the key exchange for free without modifying the parameters but the security has been reduced so much over the years that it does not seem reasonable to use it without further modifications. The signcryptext size for SETLA was computed without the symmetric cipher (since it depends on the size of the message itself and should be added to the naive construction as well [1]) and with Peikert's reconciliation which is less efficient but more compact than the one of NEWHOPE but still gives good correctness results in practice. The last column compares the gain in compactness of signcryptext when using SETLA instead of the mentioned scheme. We see that at the price of a larger public key, SETLA outperforms the naive concatenation of popular schemes by a significant margin. This is not a surprise since we only have to output a signature and the key exchange is done implicitly. The large public key comes partially from the lack of

[1] The considered naive constructions are actually KEM + signature and not directly encryption + signature.

Scheme	|sk|	|pk|	|Signcryptext|	Gain
DILITHIUM [50]+KYBER [29]	2863(=463+2400)	2560(=1472+1088)	3852(=2700+1152)	48%
QTESLA + NEWHOPE [12]	3648(=1856+1792)	4800(=2976+1824)	4896(=2720+2176)	60%
TESLA♯ [16]+KYBER	4512(=2112+2400)	4416(=3328+1088)	2768(=1616+2176)	29%
SETLA-KEX	608	6400	1972	-
GLP-Signcrypt	202	1475	1247	-

Table 5.2: Comparison between similar schemes using the naive construction. Values are given in bytes. This Table was made in early 2018, parameters of some of the schemes have been modified since then, trough rounds of the NIST project.

flexibility of RLWE which limits fast implementations to power of two cyclotomics and m as a multiple of n. Regarding speed, an update for the parameter set and an optimized implementations for all schemes involved would be needed to directly compare numbers. However, if the signcryption scheme has comparable dimension and rejection probability than the signature in the naive construction, the polynomial multiplication saved is in favor of signcryption. Furthermore, while each individual scheme uses a different public parameter for each public key, the signcryption scheme shares it accross participants. Since we saw in Chapter 3 that generation of the public parameter can be a non-negligible part of the computation, this would also play in favor of signcryption.

5.10 Conclusion

In this Chapter we presented a lattice-based signcryption scheme called SETLA. We chose a scheme of Malone-Lee as starting point and proposed two constructions both using the Bai-Galbraith signature at their cores. The first construction directly embeds a RLWE key exchange in the signature exactly as in the classical signcryption scheme while the second one uses RLWE encrypt as a key encapsulation mechanism. The KEX version seems to globally outperform the KEM version since even if it is heavier in terms of computation, this is not the main issue with lattices. We proposed a set of parameters targeting 128 bits of classical security following the reduction of Bai and Galbraith. We provided an analysis of correctness and a comparison with most recent schemes (using the naive construction) in the literature regarding signcryptext size. We also made a research oriented implementation to verify the soundness of the scheme while providing reasonable benchmarks. We conclude that it is possible to instantiate SETLA with parameters providing security, correctness and efficiency while still outperforming the naive construction of encrypt-then-sign with state-of-the-art schemes.

5.11 Thoughts and Future Works

Looking back at signcryption, I would say it was a great introduction to research for me. I discovered this less popular primitive by expanding my general knowledge about cryptography in the early stages of my thesis and it took me quite a while to put everything together but I feel like I learned a lot along the way. The main reason is that proposing a scheme requires to take care of many things such as design, sketch of security proofs,

choice of parameters and implementation. Even if all are pretty simple in this case, those tasks require some versatility.

Although important for my personal development, if would rate this result quite weak from a scientific point of view. One with good understanding of the design of lattice-based primitives and signcryption would easily find the connection between the scheme of Malonne-Lee and a NEWHOPE/ring-TESLA hybrid. Also, the proofs of security in a signcryption model are lacking, which is a bit disappointing for a signcryption scheme. Furthermore, I realized later that implementation results are not really relevant if not optimized for a specific architecture. Finally, signcryption schemes are still quite unknown and seldom studied in cryptography. Overall, everything holds and the idea is, in my opinion, very neat, but this topic is way less important than the ones exposed in the previous chapters.

For future works, I think deriving a MLWE version and, obviously, proposing a proof in a signcryption model are the two most natural directions. The reason why a module version is very much needed is actually technical. In the proof of Bai and Galbraith, it is required that $m > n$ for the underlying LWE instance. Since in RLWE, each polynomial represents exactly n samples, it is required to take at least two them (and we have $m = 2n$) for the proof to work. Module-LWE offers more flexibility on this side and would thus provide better performances. Implementation-wise, the scheme does not present any more challenge than the lattice-based signatures/KEMs it is based on and its performances are expected to follow theirs.

Chapter 6

Conclusion

This manuscript presented several practical issues faced by lattice-based cryptography, illustrated by novel research in the field. Since each chapter already concluded and discussed its main topic separately, this general conclusion will quickly summarize what has been done and give some final thoughts but will be short.

In Chapter 3, we presented new advances in fast implementations on embedded devices. In particular, we focused on polynomial multiplication techniques in $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ using specialized Fast Fourier Transform algorithms called Number Theoretic Transforms. The targeted platform was the ARM Cortex-M4 core which features SIMD instructions enabling assembly level optimizations. Beside improving speed, we also discussed several possible trade-offs in terms of code size, stack usage and bandwidth. Our results show improvements over previous work and were integrated in the current reference library for such kind of implementations. Confirming the conclusion of other researchers, after optimizing the polynomial multiplication, the run-time is vastly dominated by the random numbers generation through the hash function, which is somewhat independent of the scheme itself. While the amount of random numbers needed depends on the scheme and the parameters chosen, the need for fast random number generators is not specific to lattice-based cryptography.

In Chapter 4, we tackled the issue of side-channel attacks. We described how an attacker accessing power consumption of a device running a cryptographic algorithm might retrieve some secret data. Afterward, we discussed a countermeasure called masking which aims at splitting sensitive data in independent shares to increase the amount of information that an attacker needs to extract from the power consumption in order to learn secret values. The flagship contribution is a masking scheme for the lattice-based signature QTESLA that is proven secure in a simple yet relevant model called the probing model. The masked signature has been implemented and experimental results assessing its scaling in function of the masking order have been presented.

In Chapter 5, we discussed a lesser-known cryptographic primitive called signcryption. The goal of such a scheme is to act as both a signature and an encryption on a message more efficiently than concatenating those two basic primitives. Since signcryption was already scarcely studied in classical cryptography, it goes without saying that the post-quantum side has been barely investigated. We proposed a lattice-based signcryption scheme obtained by merging Ring Learning with Errors schemes presented in previous chapters. The scheme was supported by a brief analysis of security, correctness and efficiency.

As final thoughts, I would like to express that working on practical issues in the frame-

work of academia was pretty interesting. Quite often, it is believed that academia takes care of the theory while the industry simply makes it work in the real world. The truth is that for some issues like the ones exposed in this thesis manuscript, it is important to have people making the bridge between theoreticians and engineers. While the former sometimes brush off the performance aspect or are not well versed in coding, the latter might not grasp every subtleties of the design and make some small but harmful mistakes during implementation. When I started doing research, I was expecting a more theoretical work, but I now believe I prefer keeping a foot in the practical side. Being able to work on implementations while avoiding the inherent pressure of industry is, in my opinion, a real benefit of academia as it offers the possibility to explore many possible approaches without being too constrained by deadlines. Finally, I am really excited about the future of the NIST project as it has already spanned a lot of research and discussions on post-quantum algorithms. Having several concrete instanciations in a sort of formal competition gives many new opportunities to work on implementations, side-channel attacks, comparisons and cryptanalysis. My only complaint would be that the rigidity of the process makes it hard to tweak algorithms along the way, even if improvements or new ideas are found. For example, if the idea to create a masking friendly Fiat-Shamir lattice-based signature scheme as suggested in Chapter 4 is sometime materialized, it would likely stay outside of the process instead of being integrated as a variant of existing schemes. Naturally, I understand that this rigidity is well needed to avoid restarting analysis from scratch every time a candidate decides to incorporate major changes and I am truly looking forward to see what is going to happen in the round 3 which is, at the time those lines are written, supposed to start soon.

Bibliography

[1] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 99–108. ACM, 1996.

[2] Sedat Akleylek, Nina Bindel, Johannes A. Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An efficient lattice-based signature scheme with provably secure instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 44–60. Springer, Heidelberg, April 2016.

[3] Akshima, Donghoon Chang, Mohona Ghosh, Aarushi Goel, and Somitra Kumar Sanadhya. Single key recovery attacks on 9-round kalyna-128/256 and kalyna-256/512. In Soonhak Kwon and Aaram Yun, editors, *ICISC 15*, volume 9558 of *LNCS*, pages 119–135. Springer, Heidelberg, November 2016.

[4] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, April / May 2017.

[5] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169 – 203, 2015.

[6] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-m4 optimizations for R,M lwe schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):336–357, Jun. 2020.

[7] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qtesla. Cryptology ePrint Archive, Report 2019/085, 2019. `https://eprint.iacr.org/2019/085`.

[8] Erdem Alkim, Yusuf Alper Bilgin, and Murat Cenk. Compact and simple RLWE based key encapsulation mechanism. In *LATINCRYPT 2019*, LNCS, pages 237–256. Springer, Heidelberg, 2019.

[9] Erdem Alkim, Nina Bindel, Johannes Buchmann, and Özgür Dagdelen. TESLA: Tightly-secure efficient signatures from standard lattices. Cryptology ePrint Archive, Report 2015/755, 2015.

[10] Erdem Alkim, Nina Bindel, Johannes Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting tesla in the quantum random oracle model. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 143–162, Cham, 2017. Springer International Publishing.

[11] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016. http://eprint.iacr.org/2016/1157.

[12] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[13] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. Newhope on arm cortex-m. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 332–349. Springer, 2016.

[14] Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. In David Naccache and Pascal Paillier, editors, *PKC 2002*, volume 2274 of *LNCS*, pages 80–98. Springer, Heidelberg, February 2002.

[15] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

[16] Paulo S. L. M. Barreto, Patrick Longa, Michael Naehrig, Jefferson E. Ricardini, and Gustavo Zanon. Sharper ring-LWE signatures. Cryptology ePrint Archive, Report 2016/1026, 2016. http://eprint.iacr.org/2016/1026.

[17] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 311–323, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[18] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

[19] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

[20] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. Galactics: Gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 2147–2164, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[22] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. qTESLA. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[23] Nina Bindel, Johannes Buchmann, Juliane Krämer, Heiko Mantel, Johannes Schickel, and Alexandra Weber. Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. In Abdessamad Imine, José M. Fernandez, Jean-Yves Marion, Luigi Logrippo, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 225–241, Cham, 2018. Springer International Publishing.

[24] David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *CoRR*, abs/1805.01407, 2018.

[25] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 450–466. Springer, 2007.

[26] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011.

[27] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, October 2016.

[28] Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE, 2015.

[29] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. Crystals - kyber: A cca-secure module-lattice-based kem. *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, 2017.

[30] Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Fly, you fool! Faster frodo for the ARM cortex-M4. Cryptology ePrint Archive, Report 2018/1116, 2018. `https://eprint.iacr.org/2018/1116`.

[31] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-efficient high-speed implementation of Kyber on cortex-M4. In *AFRICACRYPT 19*, LNCS, pages 209–228. Springer, Heidelberg, 2019.

[32] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

[33] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, Heidelberg, August 2016.

[34] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR TCHES*, 2018(3):21–43, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/7267`.

[35] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[36] Eleanor Chu and Alan George. *Inside The FFT Black Box*. CRC Press, 2000.

[37] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[38] Jean-Sébastien Coron. High-order conversion from Boolean to arithmetic masking. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 93–114. Springer, Heidelberg, September 2017.

[39] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to Boolean masking with logarithmic complexity. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 130–149. Springer, Heidelberg, March 2015.

[40] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In *International Workshop on Fast Software Encryption*, pages 130–149. Springer, 2015.

[41] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Heidelberg, September 2014.

[42] R. Crandall and C.B. Pomerance. *Prime Numbers: A Computational Perspective*. Lecture notes in statistics. Springer New York, 2006.

[43] Özgür Dagdelen, Rachid El Bansarkhani, Florian Göpfert, Tim Güneysu, Tobias Oder, Thomas Pöppelmann, Ana Helena Sánchez, and Peter Schwabe. High-speed signatures from standard lattices. In Diego F. Aranha and Alfred Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 84–103. Springer, Heidelberg, September 2015.

[44] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[45] Alexander W. Dent and Yuliang Zheng. *Practical Signcryption*. Springer, 2010.

[46] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[47] Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012. http://eprint.iacr.org/2012/688.

[48] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

[49] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.

[50] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. volume 2018, pages 238–268. Ruhr-Universität Bochum, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839.

[51] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 140–158. Springer, Heidelberg, August 2016.

[52] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1857–1874. ACM Press, October / November 2017.

[53] Stephane Fernandes Medeiros, François Gérard, Nikita Veshchikov, Liran Lerman, and Olivier Markowitch. Breaking kalyna 128/128 with power attacks. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 402–414, Cham, 2016. Springer International Publishing.

[54] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

[55] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[56] Oscar Garcia-Morchon, Zhenfei Zhang, Sauvik Bhattacharya, Ronald Rietman, Ludo Tolhuizen, Jose-Luis Torre-Arce, Hayo Baan, Markku-Juhani O. Saarinen, Scott Fluhrer, Thijs Laarhoven, and Rachel Player. Round5. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[57] W. Gentleman and G. Sande. Fast fourier transforms: for fun and profit. In *AFIPS '66 (Fall)*, 1966.

[58] François Gérard, Veronika Kuchta, Rajeev Anand Sahu, Gaurav Sharma, and Olivier Markowitch. Fully homomorphic distributed identity-based encryption resilient to continual auxiliary input leakage. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications - Volume 2: SECRYPT,*, pages 41–52. INSTICC, SciTePress, 2018.

[59] François Gérard and Keno Merckx. SETLA: Signature and encryption from lattices. In Jan Camenisch and Panos Papadimitratos, editors, *CANS 18*, volume 11124 of *LNCS*, pages 299–320. Springer, Heidelberg, September / October 2018.

[60] François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qTESLA. Cryptology ePrint Archive, Report 2019/606, 2019. `https://eprint.iacr.org/2019/606`.

[61] François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qtesla. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications*, pages 74–91, Cham, 2020. Springer International Publishing.

[62] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Heidelberg, May 2001.

[63] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012.

[64] Mike Hamburg. Three Bears. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[65] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[66] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing.

[67] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[68] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[69] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4, commit e40de2b07c30aafffdf7e1a6ed086d6573f92f84. https://github.com/mupq/pqm4/commit/e40de2b07c30aafffdf7e1a6ed086d6573f92f84.

[70] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking nist pqc on arm cortex-m4. Second NIST PQC Standardization Conference, 2019. https://eprint.iacr.org/2019/844.

[71] Jonathan Katz. *Digital Signatures*. Springer US, 2010.

[72] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Heidelberg, April / May 2018.

[73] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[74] Adeline Langlois and Damien Stehlé. Hardness of decision (r)LWE for any modulus. Cryptology ePrint Archive, Report 2012/091, 2012. http://eprint.iacr.org/2012/091.

[75] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015. https://eprint.iacr.org/2012/090.

[76] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *MATH. ANN*, 261:515–534, 1982.

[77] Fagen Li, Fahad T. Bin Muhaya, Muhammad Khurram Khan, and Tsuyoshi Takagi. Lattice-based signcryption. *Concurrency and Computation: Practice and Experience*, 25(14):2112–2122, 2013.

[78] Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[79] Xiuhua Lu, Qiaoyan Wen, Zhengping Jin, Licheng Wang, and Chunli Yang. A lattice-based signcryption scheme without random oracles. *Frontiers of Computer Science*, 8(4):667–675, Aug 2014.

[80] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.

[81] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.

[82] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[83] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

[84] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.

[85] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast ntru using NTT. volume 2019, pages 180–201. Ruhr-Universität Bochum, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8293`.

[86] John Malone-Lee. Signcryption with non-interactive non-repudiation. *Designs, Codes and Cryptography*, 37(1):81–109, 2005.

[87] Robert J. McEliece. A public key cryptosystem based on algebraic coding theory. 1978.

[88] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[89] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *43rd FOCS*, pages 356–365. IEEE Computer Society Press, November 2002.

[90] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[91] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 455–485. Springer, Heidelberg, August 2017.

[92] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In *ACNS 19*, LNCS, pages 344–362. Springer, Heidelberg, 2019.

[93] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[94] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. Computational alternatives to random number generators. In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 72–80. Springer, Heidelberg, August 1999.

[95] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[96] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[97] Roman Oliynykov, Ivan Gorbenko, Viktor Dolgov, and Viktor Ruzhentsev. Results of ukrainian national public cryptographic competition. *Tatra Mountains Mathematical Publications*, 47, 12 2010.

[98] Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov, Ruslan Mordvinov, and Dmytro Kaidalov. A new encryption standard of Ukraine: The Kalyna block cipher. Cryptology ePrint Archive, Report 2015/650, 2015. `http://eprint.iacr.org/2015/650`.

[99] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

[100] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014*, pages 197–219. Springer, Heidelberg, October 2014.

[101] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10:283–424, 2016.

[102] Chris Peikert and Zachary Pepin. Algebraically structured lwe, revisited. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography*, pages 1–23. Springer International Publishing, 2019.

[103] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1843–1855. ACM Press, October / November 2017.

[104] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking deterministic signature schemes using fault attacks. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 338–352. IEEE, 2018.

[105] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. volume 13, pages 361–396. Springer, Heidelberg, June 2000.

[106] Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[107] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[108] qTESLA team. `https://qtesla.org/`.

[109] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[110] Shingo Sato and Junji Shikata. Lattice-based signcryption without random oracles. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 331–351, Cham, 2018. Springer International Publishing.

[111] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[112] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 2005.

[113] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[114] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039, 2018. `https://eprint.iacr.org/2018/039`.

[115] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[116] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `https://eprint.iacr.org/2004/332`.

[117] Yongha Son and Jung Hee Cheon. Revisiting the hybrid attack on sparse and ternary secret LWE. Cryptology ePrint Archive, Report 2019/1019, 2019. `https://eprint.iacr.org/2019/1019`.

[118] Kyber Team. Kyber resources page. https://pq-crystals.org/kyber/resources.shtml.

[119] NewHope Team. Newhope resources page. https://newhopecrypto.org/resources.shtml.

[120] Yiannis Tsiounis and Moti Yung. On the security of elgamal based encryption. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, pages 117–134, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[121] Franz Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag Wien, 1996.

[122] Jianhua Yan, Licheng Wang, Lihua Wang, Yixian Yang, and Wenbin Yao. Efficient lattice-based signcryption in standard model. *Mathematical Problems in Engineering*, 2013:1–18, 11 2013.

[123] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[124] Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 165–179. Springer, Heidelberg, August 1997.