Katrine Fjose Johansen

Dilithium, a Quantum Safe Signature

Master's thesis in Mathematical Sciences
Supervisor: Kristian Gjøsteen
June 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Kunnskap for en bedre verden

Katrine Fjose Johansen

Dilithium, a Quantum Safe Signature

Master’s thesis in Mathematical Sciences
Supervisor: Kristian Gjøsteen
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Abstract

The main purpose of this thesis is to study the Dilithium signature scheme proposed for the American National Institute of Standards and Technology's (NIST) Post-Quantum Cryptography Standardization process. In order to understand the scheme, one section is devoted to develop the necessary theory about lattices and their hard problems. We then examine identification schemes and signature schemes more generally. Afther presenting Luybashevsky and Dilithium as schemes, in addition to their security, we discus the improvements made to Dilithium compared to Lyubashevsky.

Sammendrag

Hovedformålet med denne oppgaven er å studere signatursystemet Dilithium, som ble foreslått til det amerikanske National Institute of Standards and Technology (NIST) sin “Post-Quantum Cryptography Standardization” prosess. For å forstå systemet er ett kapittel satt av til å utlese den nødvendige teorien for gittere og tilhørende vanskelige problemer. Vi ser så på den generelle struktureringen av identifikasjonssystemer og signatursystemer. Etter å ha presentert Fiat-Shamir, Luybashevsky og Dilithium som signaturer, i tillegg til deres sikkerhet, studerer vi forbedringene som er gjort med Dilithium sammenlignet med de tidligere systemene.

Acknowledgement

This thesis marks the end for my degree of Master of Science in Mathematics and the end of my time as a student at NTNU. I would like to take this opportunity to send out my gratitude to all that helped me with this thesis.

I want to thank my supervisor Kristian Gjøsteen for his suggestion of topic for this thesis. It has been a great topic to dive into and lots of fun, in addition to some frustration. Thanks to Anders and Aleksander for proofreading and to study room 393c for endless coffee breaks.

Finally, I would like to send a special thanks to my friend Aleksander Madsen Dahl for his support and for keeping me somewhat sane throughout the last year.

Katrine Fjose Johansen
Trondheim, June 2021

Abbreviations

CRH	Collision Resistant Hash.
CUR	Computational Unique Response.
CVP	Closest vector problem.
ID-Scheme	Identification schemes.
NIST	National Institute of Standards and Technology.
RO	Random Oracle.
ROM	Random Oracle Model.
SHA	Secure Hash Algorithm.
SIS	Shortest integer solution.
sk	signature key.
SUF-CMA	Strong unforgeability under chosen message attack.
SVP	Shortest vector problem.
UF-NMA	unforgeability under no-message attack.
vk	verification key.
WI	Witness Indistinguishability.
XOF	extendable-output function.
ZK	Zero-Knowledge.

Contents

List of Figures

List of Tables

1 Introduction

Modern crypto systems are mostly based on the hardness of factoring large numbers and discreet logarithms [10, 28, 30]. These problems are considered near impossible to solve using modern computers, when the numbers in use becomes very large. Therefore we considered the systems based on these problems to be safe. However, in 1994 Peter Shor [32] presented an algorithm for quantum computers that would factor large numbers much faster than the algorithms for our normal computers. Concretely, Gidney and Ekerå [15] has improved upon Shor and constructed algorithms that uses estimates of twenty million qubits to factor 2048 RSA integers. This presented a problem; our modern crypto systems will be useless, provided we are able to produce strong enough quantum computers.

But what is a quantum computer? A quantum computer uses qubits instead of normal bits, used by regular computers. These qubits can, as normal bits, be represented as 0 or 1, but also be a superposition of both. This allows us in some cases to create algorithms, such as Shor's algorithm for factoring large numbers, that are more efficient than regular algorithms on normal computers. These algorithms can again be implemented to reduce the overall time of larger computations.

How strong are quantum computers today, and are they of any real threat to us? IBM possesses, at this point in time, quantum computers of 27-qubits, and by their own scale a 64 quantum "volume" computer. This scale for measuring the power of a quantum computer considers multiple factors, such as the number of qubits and quantum noise, to compute the efficiency of the computer. The method uses many different test algorithms for benchmarking. Dash et al. [9] used a 5- and 16-qubit quantum computer to factor 4088459 and 966887 in 2018. These consist of not very large primes and are certainly not near the size of primes used in RSA, so cryptography-braking quantum computers are still some time away.

Even though quantum computers are not breaking all our systems today, the threat is still looming and we still have a wish for the safest usable crypto systems possible. So how do we protect ourselves from this threat? One option is to increase the size of the primes used to generate the private and public keys in RSA, as done by Bernstein et al. [4], but having a public key the size of 1 terabyte is rather ludicrous. Other options would be to build systems based on other problems that are not easily solved by normal nor quantum computers. Some examples of such systems are hash-based, code-based, lattice-based and multivariate cryptography.

In 2016, the American National Institute of Standards and Technology's (NIST) announced a Post-Quantum Cryptography Standardization process. NIST aim with this process to find and standardize quantum safe cryptography systems [2]. July 22, 2020 the candidates for the third round was announced and there are seven main candidates and eight alternative candidates. For signatures their three main candidates are:

- CRYSTALS-DILITHIUM
- FALCON
- Rainbow

and their three alternative candidates are:

- GeMSS
- Picnic
- SPHINCS+

So far, NIST view Dilithium [11] and Falcon [14] to be the most promising signature schemes out of the candidates. Both of these signature schemes base their security on the hardness of lattice problems. Rainbow [33] is a multivariate cryptosystem and was first constructed roughly fifteen years ago.

In this thesis we will study Dilithium as a cryptosystem. We start by looking at the underlying lattice structures and problems in section 2. This understanding make it possible for us to construct systems that are secure against quantum computers. As Dilithium is a signature scheme, we will present what signatures are and what security properties they have in section 3. In section 4 we are going to present our signature schemes, Lyubashevsky and Dilithium, whom are both quantum safe. Lastly, in section 5, we will discuss the differences of these schemes and how they build upon one another to arrive at a secure and efficient signature scheme suitable for standardisation.

2 Preliminary

2.1 Lattices

When discussing cryptographic systems that are secure against quantum computers we have many options, including, but not limited to, lattice-based systems, isogeny-based systems and multivariant-based systems. The most promising to date are lattice-based crypto systems. There are many and complex reasons for this, but we are not going to focus on them. In this section we will describe what lattices are and why they are secure for our use. The definitions in this section are standardised and we follow Gjøsteen, Lyubashevsky and Laarhoven et al. [16, 17, 22, 25].

Definition 2.1 *Let $\mathbb{R}^n$ be the n-dimensional vector space over $\mathbb{R}$ and let $B = \{\boldsymbol{v}_1, ..., \boldsymbol{v}_n\}$ be a basis for $\mathbb{R}^n$. A* ***lattice*** *Λ in $\mathbb{R}^n$ generated by B is*

$$\Lambda_B = \Big\{ \sum_{i=1}^{n} a_i \boldsymbol{v}_i \mid a_i \in \mathbb{Z} \, , \; \boldsymbol{v}_i \in B \Big\}$$

We see that a lattice is a subgroup of $\mathbb{R}^n$, where its elements are integer linear combinations of a basis in $\mathbb{R}^n$. As $\mathbb{Z}^n$ is a subgroup of $\mathbb{R}^n$ there are also lattices that are subgroups of $\mathbb{Z}^n$. There exist many different types of lattices, some more interesting than others.

Definition 2.2 *An* ***integer lattice*** *Λ_B is such that $\Lambda \subseteq \mathbb{Z}^n$. In particular, $B = \{\boldsymbol{v}_1, ..., \boldsymbol{v}_n\} \subseteq \mathbb{Z}^n$.*

Definition 2.3 *Given a vector $\boldsymbol{v} = (v_1, ..., v_n)$, we define $rot(\boldsymbol{v}) = (v_n, v_1, ..., v_{n-1})$ to be the rotational shift of $\boldsymbol{v}$. A* ***cyclic lattice*** *is a lattice Λ where $\forall \boldsymbol{v} \in \Lambda \rightarrow rot(\boldsymbol{v}) \in \Lambda$.*

Examples of cyclic lattices are $\mathbb{Z}^n$ and any lattice corresponding to an ideal of $\mathbb{R}[x]/\langle x^n + 1 \rangle$. We often think of polynomials and vectors as "the same". What we mean by this is that we have a mapping from $\mathbb{F}[x]/\langle f(x) \rangle$, with $f(x)$ is a polynomial in $\mathbb{F}[x]$ of degree n, into $\mathbb{F}^n$. This is done using the following isomorphism:

$$M : \mathbb{F}[x]/\langle f(x)\rangle \to \mathbb{F}^n$$
$$v_0 + v_1x + v_2x^2 + ... + v_{n-1}x^{n-1} \mapsto (v_0, v_1, ..., v_{n-1})$$

A lattices corresponding to an ideal I in a quotient ring $\mathbb{F}[x]/\langle f(x)\rangle$ is thus the lattice whose elements can be mapped to the polynomials in the ideal I by our isomorphism M above. We will often talk about vectors and polynomials interchangeably. When we do so, we are simply using this isomorphism without necessarily mentioning it.

Definition 2.4 *Let $f(x) \in \mathbb{Z}[x]$ be a monic polynomial of degree n and let I be an ideal in the quotient ring $\mathbb{Z}[x]/\langle f(x)\rangle$. An **ideal lattice** is a cyclic lattice $\Lambda(I) \subseteq \mathbb{Z}^n$ such that its basis is $B = \{g(x)\ mod\ f(x) \mid g(x) \in I\}$ for some monic polynomial $f(x)$ and ideal I as above.*

This type of lattice is of specific interest to us, as it is exceptionally nice for the purpose of quantum safe cryptography. Particularly the lattices corresponding to ideals of $\mathbb{Z}[x]/\langle x + 1\rangle$ and $\mathbb{Z}_p[x]/\langle x + 1\rangle$, for some prime p, is a uniquely good lattice [25]. Therefore we will denote these rings as R and R_p, respectively, for the duration of this text.

Definition 2.5 *Let $p\mathbb{Z}^n = \{p\boldsymbol{v} \mid \boldsymbol{v} \in \mathbb{Z}^n\}$, we say that a lattice Λ is **p-ary** if $p\mathbb{Z}^n \subseteq \Lambda \subseteq \mathbb{Z}^n$.*

Definition 2.6 *Given some matrix $\boldsymbol{M} \in \mathbb{F}_p^{n\times m}$ we construct lattices using the matrix such that*

$$\Lambda_p(\boldsymbol{M}) = \{u \in \mathbb{Z}^n \mid \exists a, a\boldsymbol{M} \equiv u(mod\ p)\}$$
$$\Lambda_p^{\perp}(\boldsymbol{M}) = \{u \in \mathbb{Z}^n \mid \boldsymbol{M}u \equiv 0(mod\ p)\}$$

We know that if a lattice Λ is p-ary then there exist matrixes $\mathbf{M}$ and $\mathbf{M}'$ such that $\Lambda = \Lambda_p(\mathbf{M})$ and $\Lambda = \Lambda_p^{\perp}(\mathbf{M}')$.

2.2 Lattice Problems

Now that we have looked at what lattices are, we want to know what property, or rather problems, they have that makes them suitable for our use in cryptography. We will not yet show why these problems make lattices suitable, but we will present the problems.

To express the problems we need a notation for the length of a vector. The length of a vector is determined by its norm, denoted $||\mathbf{v}||$. We are mostly focusing on the infinity norm, but will use other norms occasionally. This we will be distinguished by $||\mathbf{v}||_\infty$ or $||\mathbf{v}||_g$ when relevant. We will also denote the shortest vector in a lattice Λ by $\lambda(\Lambda)$.

2.2.1 Shortest Vector Problem

Definition 2.7 *The **shortest vector problem** (SVP) for a lattice Λ is to find a nonzero vector $\boldsymbol{x} \in \Lambda$ such that $\forall$ nonzero $\boldsymbol{y} \in \Lambda$ $||\boldsymbol{x}|| \leq ||\boldsymbol{y}||$.*

It is worth noticing that SVP asks for a shortest vector and not the shortest vector. This means there could be several vectors with the same shortest norm. For instance the lattice with $\{(0,1),(1,0)\}$ as its basis has both the vector $(0,1)$ and $(1,0)$ as its shortest nonzero vectors. This is easy to see as both vectors have the norm equal to 1.

There are more than one way to formulate this problem. The above definition define the problem of finding an absolute shortest nonzero vector. However we may also want to find a vector whose length is merely shorter then γ times the length of the shortest vector. This problem is called γ-approximate SVP.

Definition 2.8 *The **γ-approximate shortest vector problem** (SVP_γ) for a lattice Λ is to find a nonzero vector $\boldsymbol{x} \in \Lambda$ such that $\forall$ nonzero $\boldsymbol{y} \in \Lambda$ $||\boldsymbol{x}|| \leq \gamma||\boldsymbol{y}||$.*

We know how to solve these problems for general or “good” lattices. Looking back at our previous example, finding a shortest vector was extremely easy as the basis is very nice. A well known algorithm for finding short vectors in lattices is the LLL-algorithm, developed by Lenstra, Lenstra and Lovász [23]. The algorithm takes in a basis and puts out a LLL-basis. This basis will then contain a short vector, and solve SVP$_\gamma$, but γ grows exponentially by the dimension of the lattice. This algorithm can thus be used to solve SVP for lattices with a so-called "good" basis. However as the basis gets increasingly worse the algorithm gets less efficient. It has been shown by Ajtai [1] that solving SVP is NP-hard and later likewise by Khot [20] for the approximate SVP.

Definition 2.9 *Given a basis of Λ and an approximation factor $\gamma > 1$, the* ***Shortest Independent Vector Problem*** *($SIVP_\gamma$) is to find a linearly independent set $\{y_1, ..., y_d\}$ such that max_i $||y_i|| \leq \gamma\lambda_d(\Lambda)$.*

Blömer and Seifert [7] showed that SIVP_γ is NP-hard.

2.2.2 Closest Vector Problem

Definition 2.10 *The* ***closest vector problem*** *(CVP) for a lattice Λ and a vector $\boldsymbol{z} \in \mathbb{R}^n$ is to find a nonzero vector $\boldsymbol{x} \in \Lambda$ such that $\forall$ nonzero $\boldsymbol{y} \in \Lambda$ $||\boldsymbol{z} - \boldsymbol{x}|| \leq ||\boldsymbol{z} - \boldsymbol{y}||$.*

If we want to we can formulate SVP as a CVP simply by setting our vector $\mathbf{z} \in \mathbb{R}^n$ to be the origin. Similarly to SVP, we don't have to define the problem to find the absolute closest vector. We can also construct the problem to merely find the closest one within a given radius, by expanding the problem to a CVP_γ problem.

Definition 2.11 *The* ***γ-approximate closest vector problem*** *(CVP_γ) for a lattice Λ and a vector $\boldsymbol{z} \in \mathbb{R}^n$ is to find a nonzero vector $\boldsymbol{x} \in \Lambda$ such that $\forall$ nonzero $\boldsymbol{y} \in \Lambda$ $||\boldsymbol{z} - \boldsymbol{x}|| \leq \gamma||\boldsymbol{z} - \boldsymbol{y}||$.*

Because of the similarities between SVP and CVP, solving CVP would also give us a solution for SVP. One approach for solving CVP has been to round every element of the vector to match the nearest lattice vector. However this is not an especially satisfying method if our lattice is not constructed with a very "good" basis.

2.2.3 Short Integer Solution

Definition 2.12 *The* ***Short integer solution*** *(SIS) for $\Lambda(\boldsymbol{M})$ of a given $n \times m$ matrix $\boldsymbol{M} \leftarrow \mathbb{Z}_p^{n\times m}$ is finding a nonzero vector $\boldsymbol{x} \in \mathbb{Z}^m$ such that $\boldsymbol{Mx} = 0$ and*

$||\boldsymbol{x}|| \leq \beta$.

SIS is a problem related to lattices that are constructed by matrices. Here we wish to find a vector such that our matrix times this vector computes to the zero-vector, while also having a norm smaller than β. The smaller β is the harder this problem gets. Therefore we often wish to put a lower restriction on β, so as to not make the problem impossible. Lyubashevsky [25] restrict this parameter by $\beta \leq \sqrt{m}p^{n/m}$, and denote the problem with the parameters by $\mathrm{SIS}_{n,m,q,\beta}$.

2.2.4 Learning With Error

Definition 2.13 *Given n, q, χ and any number (m) of independent samples from $A_{s,\chi}$, the **Learning With Errors** ($LWE_{n,m,q\chi}$) problem is to find s.*

Not all the problems related to lattices are as similar to each other at first sight, but they are all connected to each other. Laarhoven, Pol and Werger [22] present a figure to visualise the connections between the problems. We have taken inspiration from them and construct a similar figure with the lattice problems presented in this subsection in Figure 1.

$$\begin{tikzcd}
CVP_\gamma & SVP_\gamma \arrow[l, "{[18]}"'] & SIVP_\gamma \arrow[l, "{[19]}"'] \arrow[r, "{[26]}"] \arrow[rr, bend right] & SIS_{n,m,q,\beta} \arrow[ll, bend right] \arrow[r, bend left, "{[34]}"] & LWE_{n,m,q,bl\chi} \arrow[l, bend left, "{[27]}"]
\end{tikzcd}$$

Figure 1: Reduction of Lattice Problems

2.3 Hash Functions

We will now have a look at some different types of hash functions, before connecting the security to our lattices. Cryptographic hash functions are binary functions designed to provide collision resistance and preimage resistance, among other properties. These functions take bit-strings, called messages, as input, and output what we call hash values. We start by looking at unstructured hash functions and then move on to structured hash functions.

2.3.1 Unstructured hash function

Secure hash algorithm also known as SHA, are groups of cryptographic hash functions that have been certified for use in cryptographic systems by NIST. These kinds of hash functions are constructed to appear completely random, and should not have any internal structures. This makes it near impossible to compute interchangeably between messages and hash values. There are three main classes of SHA, namely SHA1, SHA2 and SHA3. SHA3 is the most recent class and all six functions are based on KECCAK [6], the winner of NISTs cryptographic hash algorithm competition [8]. The six functions are categorised into two types of functions, hash functions and extendable-output functions.

Hash functions are constructed to be collision resistant and appear like a random one to one function. These functions go from some large domain to a fixed smaller domain. Within crypto, the main purpose of these functions are to hide big messages by making them small and appear completely random, while making sure it is near impossible to find a collision. Often they are described as collision resistant hash or CRH for short. The four hash functions in SHA3 are SHA3-224, SHA3-256, SHA3-384 and SHA3-512

Contrary to hash functions, extendable-output functions, XOF, does not have a fixed output size and the output is mostly extended, hence the name. Except for this, XOF are constructed mostly simular to the other hash functions and thus also have the same demands for randomness and to be unstructured. The first two extendable-output functions, standardised by NIST [31], are SHAKE-128 and SHAKE-256.

Given a message μ, we can use KECCAK to present the SHA3 hash functions as follows:

$$\begin{aligned}\text{SHA3-224}(\mu) &= \text{KECCAK}[448](\mu||01, 224);\\ \text{SHA3-256}(\mu) &= \text{KECCAK}[512](\mu||01, 256);\\ \text{SHA3-384}(\mu) &= \text{KECCAK}[768](\mu||01, 384);\\ \text{SHA3-512}(\mu) &= \text{KECCAK}[1024](\mu||01, 512).\end{aligned}$$

We can also present the XOF of SHA3 still using the KECCAK algorithm with d as its declared output size.

$$\begin{aligned}\text{SHAKE128}(\mu, d) &= \text{KECCAK}[256](\mu||1111, d);\\ \text{SHAKE256}(\mu, d) &= \text{KECCAK}[256](\mu||1111, d).\end{aligned}$$

Function	Output Size	Collision	Preimage	2and Preimage
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE128	d	min(d/2,128)	≤min(d,128)	min(d,128)
SHAKE256	d	min(d/2,256)	≤min(d,256)	min(d,256)

Table 1: Security strength of SHA3[31]

As we can see both, hash functions and XOF are constructed very similarly, and one can make an XOF into a hash function by simply choosing its output size to match that of the hash functions. The security properties for the functions in SHA3 is inherent from KECCAK and the sponge constructions, which was proven by Bertoni et al. [5, 6]. The interested reader may have a look at these, but as it is not the main focus of this thesis we are not going into details here. The security strengths of the functions can however be seen in Table 1.

2.3.2 Structured hash function

Sometimes we want random seeming functions with a bit more structure. This is so that we may compute interchangeably between the message and the hash value. Recall that R is given by the ring $\mathbb{Z}[x]/\langle x^n+1\rangle$ and likewise R_p by $\mathbb{Z}_p[x]/\langle x^n+1\rangle$.

Definition 2.14 *Given any integer m and let $D \subseteq S$, we define a* ***hash function family*** *$H(S, D, m)$ to be the set of maps from D^m to S such that*

$$H(S, D, m) = \{h_{\boldsymbol{a}} \mid \boldsymbol{a} \in S^m\} \text{ where for any } \boldsymbol{z} \in D^m, h_{\boldsymbol{a}}(\boldsymbol{z}) = \boldsymbol{a} \cdot \boldsymbol{z}$$

It is easy to see that these hash functions have a linear structure. This might seem as though it would weaken their security, but we will prove that, when constructed properly, they are secure for our intended use. An important property, as mentioned above, of hash functions is the collision resistance.

Definition 2.15 *Given $h \in H(S, D, m)$, the* ***collision problem****, Col(h), asks to find two elements $\boldsymbol{z}, \boldsymbol{z'} \in D$ such that $h(\boldsymbol{z}) = h(\boldsymbol{z'})$*

When D is a restricted domain in R it can be shown that solving the collision problem for hash function families is as hard as solving the SVP$_\gamma$ for any (x^n+1)-cyclic lattice. Thus finding a collision in a hash function family for R_p will give us a solution to the SVP$_\gamma$ for an ideal lattice $\subseteq R_p$. We will prove this statement by the propositions in this section. Our proofs follow the proof by Lyubashevsky and Micciancio [24]. Before stating our propositions, we will present some algorithms, namely AL, CollFind and RAL.

AL takes $\Lambda(I), m, \gamma$ and g as input:

```
(1)       $s = \frac{||g||_\infty}{16dm\sqrt{n}\log(n)} \geq \frac{16dmn\log^2(n)\lambda(\Lambda)}{16dm\sqrt{n}\log(n)} \geq \sqrt{n}\log(n)\lambda(I)$
(2)       for $i = 1$ to $m$
   (2.1)       generate a uniformly random coset of $I/\langle g\rangle$ and let $v_i$ be a polynomial in $I/\langle g\rangle$.
   (2.2)       generate $y_i \in \mathbb{R}^n$ such that $y_i$ has distribution $\rho_s/s^n$ and consider $y_i$ as a polynomial in $\mathbb{R}[x]$.
   (2.3)       let $w_i$ be the unique polynomial in $\mathbb{R}[x]$ with degree less than n and coefficients in the range $[0, p)$ such that
               $p(v_i + y_i) = gw_i$ in $\mathbb{R}^n/\langle pg\rangle$.
   (2.4)       $a_i = [w_i] \bmod p$, here $[w_i]$ rounds each coefficient to the closest integer.
(3) output $\mathbf{a}, \mathbf{w}$ and $\mathbf{y}$
```

CollFind takes $g, \mathbf{a}, \mathbf{w}$ and $\mathbf{y}$ as input:

```
(1)       calls an oracle $\mathcal{C}(a_1, ...a_m)$ that outputs $\alpha$ and $\beta$ such that
          $||\alpha||_f, ||\beta||_f \leq d$ and $\sum a_i\alpha_i = \sum a_i\beta_i$
(2)       for $i = 1$ to $m$
   (2.1)       $z_i = \alpha_i - \beta_i$ such that $||z_i||_f \leq 2d$ and $\sum a_i z_i \equiv 0$ in $R_p$
(3)       output $\mathbf{z}$
```

RAL takes $\Lambda(I), g, \mathbf{w}, \mathbf{y}$ and $\mathbf{z}$ as input:

```
(1)       $l = \left(\sum\left(\frac{g(w_i - [w_i])}{p} - y_i\right)z_i\right) \bmod (x^n + 1)$
(2)       output $l$
```

Proposition 2.1 *Let $\Lambda(I)$ be an ideal lattice, where I is an ideal in R_p, with $m > \frac{\log p}{\log 2d}$, $p \geq 4dmn^{1.5}\log(n)$, $\gamma = 16dmn\log^2(n)$ and $D = \{\boldsymbol{y} \in R_p \mid ||\boldsymbol{y}||_\infty \leq d\}$ for some d. When the algorithm AL takes in $\Lambda(I)$ together with R_p and the parameters above it puts out a vector $\boldsymbol{a}$, of polynomials a_i, such that a_i are statistically close to uniformly and independently random distributed.*

Proof Before using the algorithm AL we need a $g \in I$, I ideal of R_p, such that $g \neq 0$ and $||g||_f \geq \gamma\lambda(I)$, where $f = x^n + 1$. We know that I has finite dimension as R_p has finite dimension and we set $D = \{k \in R_p \mid ||k||_f < d\}$. In addition we assume that g is of less degree than n and thus we know that $||g||_f = ||g||_\infty$. We then run AL with $\Lambda(\lambda), m, \gamma$ and g as its input.

Since we generated $I/\langle g\rangle$ to be a uniformly random coset, v_i is then in a uniformly random coset. Let us assume that y_i is also from a uniformly random coset, then $v_i + y_i$ will also be. Then the distribution of $p(v_i + y_i)$ is from a uniformly random coset of $\mathbb{R}/\langle pg\rangle$. A basis for $\langle pg\rangle$ can be of the form $pg, pgx, ..., pgx^{n-1}$, thus an element in $\mathbb{R}^n/\langle pg\rangle$ can be presented as $\alpha_0 pg, \alpha_1 pgx, ..., \alpha_{n-1}pgx^{n-1} = p(\alpha_0 g, \alpha_1 gx, ..., \alpha_{n-1}gx^{n-1})$ for some $\alpha_i \in [0, 1)$. We can write $w_i = p\alpha_0 + p\alpha_1 x + ... + p\alpha_{n-1}x^{n-1}$ and since $p(v_i + y_i)$ is in a uniformly random coset it follows that w_i is uniform over $[0, p)$. This makes the coefficients in $[w_i]$ uniform over the integers modulo p.

However, since we only assumed that y_i is from a uniformly random coset, we are not done. The distribution of y_i is ρ_s/s^n and so by our choice of s we have that $\Delta(\rho_s/s^n + I, U(\mathbb{R}^n/I)) \leq \epsilon/2$. Using equation 1 and 2 [24] together with the fact that a_i's are functions of y_i and independent we get that $\Delta((a_1, ..., a_m),\ U(\mathbb{Z}_p^{n\times m})) \leq m\epsilon/2$. This means that our a_i is statistically close to uniform, and this is good enough.

□

Having a vector constructed by the algorithm AL from Proposition 2.1 we can construct a hash function from $H(R, D, m)$. Having this hash we can now use our algorithm CollFind to find a collision in our hash. The algorithm uses an oracle to find polynomials $\alpha_1, ..., \alpha_m, \beta_1, ..., \beta_m$ such that $||\alpha||_f, ||\beta||_f \leq d$ and $\sum a_i\alpha_i = \sum a_i\beta_i$. Then it sets $z_i = \alpha_i - \beta_i$ and output $z_1, ..., z_m$ such that $||z_i||_f \leq 2d$ and $\sum z_i a_i \equiv 0$ in R. We now want to show that having a collision in our hash, we can use this to solve SVP$_\gamma$.

Proposition 2.2 *Let $\boldsymbol{a}, \boldsymbol{w}$ and $\boldsymbol{y}$ be the output from $AL(m, \gamma, g)$ and let $h \in H(R, D, m)$ be constructed using $\boldsymbol{a}$. If we have a collision in h, given by CollFind, then algorithm RAL gives us a solution for $SVP_\gamma(\Lambda(I))$ for every (x^n+1)-cyclic lattice $\Lambda(I)$.*

Proof Assuming we have all the output from the algorithm AL, and the collision from our algorithm CollFind we use algorithm RAL and get

$$l = \left(\sum \left(\frac{g(w_i - [w_i])}{p} - y_i\right) z_i\right) \bmod (x^n + 1)$$

To make sure that l is indeed a solution we need to show three things; that $l \in I$, that $||l||_f \leq ||g||_\infty / 2$ and that $l \neq 0$. We will separate these into three lemmas to keep our head in the game while proving them. Our proposition then follows directly from these three lemmas.

Lemma 2.1 *$l \in I$.*

Proof

$$\begin{aligned}\left(\sum \left(\frac{g(w_i - [w_i])}{p} - y_i\right) z_i\right) &= \sum (v_i + y_i + gk_i - ga_i/p - y_i) z_i \\ &= \sum (v_i + gk_i) z_i - \frac{g \sum a_i z_i}{p}\end{aligned}$$

We know that $v_i \in I$ and $g \in I$, therefore $v_i + gk_i \in I$ and it then follows that $\sum (v_i + gk_i) z_i \in I$. $\sum a_i z_i \equiv 0 \bmod p$, thus $\frac{\sum a_i z_i}{p} \in \mathbb{Z}[x]$ and $\frac{g \sum a_i z_i}{p} \in I$. Thus we have shown that $l \in I$.

□

Lemma 2.2 *$||l||_f \leq ||g||_\infty / 2$.*

Proof We start by looking at the infinity norm of l

$$\begin{aligned}||l||_\infty &= \left|\left|\sum \left(\frac{g(w_i - [w_i])}{p} - y_i\right) z_i\right|\right|_\infty \\ &\leq \sum \left|\left|\left(\frac{g(w_i - [w_i])}{p}\right) z_i\right|\right|_\infty + \sum ||y_i z_i||_\infty\end{aligned}$$

We have separated the norm into two sums and will look closer at these separately. We start with the first sum.

$$\left|\left|\left(\frac{g(w_i - [w_i])}{p}\right) z_i\right|\right|_\infty \leq \frac{1}{p}||g(w_i - [w_i])||_\infty ||z_i||_1$$

Assuming that the coefficients in w_i are uniformly distributed in $[0, p)$ independently, the coalitions in $w_i - [w_i]$ are uniformly distributed in $[-1/2, 1/2]$. Since w_i are independent from g, we get from Lemma E.2 [24] that

$$||g(w_i - [w_i])|| \leq \omega(\sqrt{n}\log(n))||g||$$

with probability negligible close to 1. We have already argued that the coefficients in w_i are statistically close to uniform over $[0, p^n)$, and the inequality therefore still holds after removing our assumption about w_i. Thus

$$\sum \left|\left|\left(\frac{g(w_i - [w_i])}{p}\right) z_i\right|\right|_\infty < \frac{||g||_\infty}{4\mathcal{E}}$$

follows from our choice of p.

For the second sum, our approach will be slightly different. We start by wanting to show that the following probability is very small.

$$\Pr[||y_i z_i||_\infty > ||z_i||_\infty s\sqrt{n}\log n \mid (a_1, ..., a_m), (z_1, ..., z_m)]$$

This is to argue that the probability that $||y_i z_i||_\infty < \frac{||g||_\infty}{4\mathcal{E}}$ is negligible close to 1. We know that for any coset of $\mathbb{R}^n/I$, for instant $y_i' + I$, the distribution of a_i is the same regardless of y_i and vice versa. The y_i's are thus also independent of $z_1, ..., z_m$, since $\mathbf{z}$ is a function of $\mathbf{a}$, and we get the following probability

$$\Pr[y_i \mid y_i \in y_i' + I] = \frac{\rho_s(y_i)}{\rho_s(y_i' + I)} = \frac{\rho_{s,-y_i'}(y_i - y_i')}{\rho_{s,-y_i'}(I)}$$

Using this probability together with Lemma 2.7 [24] we get

$$\begin{aligned}
&\Pr_{y_i \sim \rho_s/s^n}\left[||y_i z_i||_\infty > ||z_i||_\infty s\sqrt{n}\log(n) \,\middle|\, y_i \in y_i' + I\right] \\
&= \Pr_{(y_i - y_i') \sim D_{I,s,-y_i'}}\left[||((y_i - y_i') - (-y_i'))z_i||_\infty \geq ||z_i||_\infty s\sqrt{n}\log(n)\right] \\
&= n^{-\omega(1)}
\end{aligned}$$

This probability is for each i independently, and summing up all the probabilities gives us

$$\Pr\left[\sum ||y_i z_i||_\infty \geq 2dms\sqrt{n}\log(n)\right] = n^{-\omega(1)}$$

We see with probability negligible close to 1 that

$$||l||_\infty < \frac{||g||_\infty}{4\mathcal{E}} + \frac{||g||_\infty}{4\mathcal{E}} = \frac{||g||_\infty}{2\mathcal{E}}$$

and know by the constructed l that it has degree less than $3(n-1)$, thus

$$||l||_f \leq \mathcal{E}||l||_\infty \leq \frac{||g||_\infty}{2}$$

We see that we almost always get an l that is the smallest vector in our lattice. $\square$

Lemma 2.3 $l \neq 0$

Proof It is good enough that the probability that $l = 0$ is very small. Assuming without loss of generality that z_1 is non-zero, then $l = 0$ if and only if

$$y_1 z_1 = \sum_{i=1}^{m} \frac{g(w_i - [w_i])z_i}{p} - \sum_{i=2}^{m} y_i z_i$$

We know that given a coset of $\mathbb{R}^n/I$, then y_1 is independent from a_i, z_i and $y_{i>1}$. We want the bound of the probability to be as follows

$$\Pr_{y_1 \sim \rho_s/s^n}\left[y_1 z_1 = \sum_{i=1}^{m} \frac{g(w_i - [w_i])z_i}{p} - \sum_{i=2}^{m} y_i z_i \,\middle|\, y_1 \in y_1' + I \right]$$

If $z_1 y_1 = c$, then y_1 can only be one value for each given z_1, because $z_1, z_1 x, .., z_1 x^{n-1}$ are linearly independent. From equation 5 [24] it follows that

$$\Pr_{y_1 \sim \rho_s/s^n}[y_1 \mid y_1 \in y_1' + I] = \frac{\rho_s(y_1)}{\rho_s(y_1' + I)} = \frac{\rho_{s,-y_i'}(y_i - y_i')}{\rho_{s,-y_i'}(I)}$$

This is the probability that given $x \sim D_{I,s,-y_1'}$ we have $x = y_1 - y_1'$. It then follows from lemma 2.5 [24] that we have the probability $\Omega(1)$ for $l \neq 0$, which is small enough.

$\square$

Together, Proposition 2.1 and 2.2 proves that if we have a collision finder for a hash function, we get a solver for ideal SVP$_\gamma$. It is worth noting that our algorithm from Proposition 2.1 only gives us the **a** we want most of the time. However, this is fine, since it also runs in polynomial time, and if it fails we can simply run it again. It is fairly easy to show that it runs in polynomial time, thus we won't be bothering to do it in this text. The specially interested reader can have a look in the original proof [24]. It follows from this proof that since we believe it is very hard to solve SVP we can then conclude that it is equally hard to find a collision in our hash. This is great since this gives us a way to construct hash functions with our desired structures to use in quantum safe crypto systems.

3 Identification Scheme and Signatures

Before we start to look at some actual schemes we want to know what their purpose is and what security criteria we put on them. In this section we will therefore introduce the concept of identification schemes, abbreviated ID-Scheme, and signature schemes. The main purpose of such schemes is for one to be able to prove once identity or in some way to sign a message. We are going to discuss how ID-schemes and signatures are constructed and what properties they have to deem them secure. It is natural to start with ID-schemes, as signatures often are constructed with an ID-scheme in mind.

3.1 What is an Identification-Scheme?

An ID-scheme consists of two parts, a key-generator algorithm and an interactive protocol. The latter is between a prover, often called Peggy, and a verifyer, often called Victor. The key-generator algorithm generates a key par, sk and vk, that gets distributed to Peggy and Victor, respectively. Although Peggy is not necessarily signing anything we are going to call her key the signature key, or sk for short. We will call Victors key the verification key, or vk, as Victors goal is to verify. Upon receiving the keys, Peggy and Victor can run the interactive protocol between themselves.

The interactive protocol starts by having Peggy pick a random value, send it to Victor, then wait for a challeng from him. Victor selects a random challenge and sends it back to Peggy upon receiving her random value. These two values are both random and independent. Upon receiving the challenge Peggy computes a new value based on her sk and the two random values. This computation is done so that she can prove to Victor that she does indeed know the sk without revealing it to him. Lastly, Victor finishes the interaction by uses his vk, the random values and Peggys answer to determine if he believes that Peggy knows sk. He then accepts or rejects her based on whether or not he believes her. We say that our identification protocol was successful if Victor accepts Peggy.

To give a more visual idea of the interactive protocol of an ID-scheme we have included an illustration of a sigma protocol in Figure 2. In this figure we have denoted the different values being sent back and forth by α, β and γ. The specific computations of these values varies from scheme to scheme and we will look at specific ID-scheme in the next section.

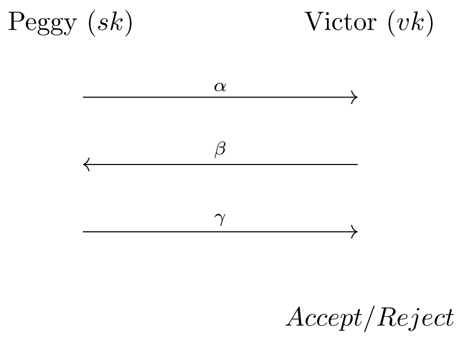

Figure 2: Sigma Protocol

3.1.1 Completeness

As much as we want a interactive protocol to succeed between two honest parties, this is not always the case, nor do we need our protocol to succeed every time. We just need the probability of acceptance to be higher than the probability of rejection when both parties are honest.

Definition 3.1 *If Peggy has sk, having* ***completeness*** *means Victor rejects with negligible probability.*

If a protocol does not have completeness it is to no use for us, as it will not prove anything. Therefore completeness is our first goal in constructing an ID-scheme. Having completeness does however only confirm Peggys ability to convince Victor to accept. It does not make sure a dishonest Peggy can not convince Victor of the same. Nor does it make sure Victor does not learn any information of sk.

3.1.2 Soundness

In a perfect world we would not have to take precautions to avoid any malicious third parties. But if the world had been perfect to begin with, Peggy would probably not have to prove herself to Victor in the first place. In the real world however, there are plenty of malicious people that want to interfere with us. Such a malicious person could for example pretend to be Peggy herself.

Definition 3.2 *A **cheating Peggy**, denoted P^*, is an interactive algorithm with access to vk, it runs the protocol with Victor such that Victor eventually terminates.*

When we say that Peggy is dishonest, malicious or cheating, we mean that she does not have access to the *sk*. If she does not know *sk* she should not be able to run a successful identification protocol with Victor.

Definition 3.3 *P^* can observe conversations between P and V and from this gets a view. This view consists of all the parameters sent between P and V. Then P^* tries to run the protocol with V and we call the probability of success for soundness error*

$$Pr[V \to 1|P^*] = \epsilon$$

We say the protocol has soundness if ϵ is negligible in our protocol.

Knowing this probability gives us the opportunity to make our schemes secure enough for our purposes. Optimally we want our ID-scheme to never accept a malicious party, however we see that in practice that this is both nearly impossible and unnecessary. Having the probability that an dishonest Peggy getting accepted, ϵ, low will make our protocol secure enough.

3.1.3 Zero-Knowledge

A malicious party may also try to get some insight on the *sk* from a honest Peggy. In this case the malicious party will take the role of a dishonest or cheating Victor.

Definition 3.4 *A **cheating Victor**, V^*, is an interactive algorithm that takes in the verification key, interact with Peggy and eventually terminates with some output.*

After our interactive algorithm has run the protocol with a honest Peggy it will be left with all the parameters that Peggy sent him. We call this collection of parameters a view. The view is interesting to us as it tells us what information the cheating Victor has gained.

Definition 3.5 *A **simulator** for Victor is an algorithm that when being fed the verification key eventually terminates with some output or a special symbol $\perp$. The probability that it outputs $\perp$ is independent from the verification key.*

The simulator will as our interactive algorithm be left with a view. To determined the security of our scheme we want to compare the view of our dishonest Victor and our simulated Victor.

Definition 3.6 *We say that we have **Zero-Knowledge** (ZK) if the view from the dishonest Victor and the view from the simulator are the same.*

We call it zero-knowledge because we want Victor to gain precisely zero knowledge about Peggys sk. If the dishonest Victor cannot gain any more information than the simulated Victor, and we have constructed our protocol right, Victor, dishonest or not, should not learn anything about sk, from the view.

3.1.4 Witness Indistinguishability

Often in ID-schemes it is possible to construct multiple sk for a single vk. This may sound like it would cause some big problems, however, often it is not so. We do have to keep it in mind when constructing ID-schemes, though. Feige and Shamir [12] define the notion of witness indestinguishability (WI) as

Definition 3.7 *A ID-scheme is witness indistinguishable over $\mathcal{R}$ if for any V^*, any large enough input μ and any two signature keys sk, sk' for a verification key vk, the views generated by running $V^*(P(\mu, sk), vk)$ and $V^*(P(\mu, sk'), vk)$ are indistinguishable for V^*.*

As with ZK, WI is about Victor's view, but here we do not need to include a simulator. In ZK we want to make sure that Victor does not learn anything from the response, whereas in WI we want Victor to not be able to see any difference between two sk corresponding to the same pk. If we make sure that our system does not leak any information about which sk we use, it will look as though there is only one sk for each vk and we need not worry much.

3.1.5 Abort

Occasionally in an ID-scheme we stumble upon the problem that Peggy can not produce a response using Victors challenge, or that by doing so the response gives out more information than is wanted. To deal with this kind of problem, Peggy can simply send a new commitment and ask for a different challenge. This procedure is called an abort.

If Peggy aborts the procedure many times in a row this might make Victor suspicious. However an honest Peggy does not need to abort a suspicious amount of times, so this is of very little concern to us.

A nice consequence about using aborting in our schemes is that we are more free to choose the size of our final response. Although having to ask for a new challenge is time consuming, so for ID-schemes this is not such a big advantage. However further development of ID-schemes have great benefits from a shorter response, as we will see later. Having shorter responses makes it both more efficient to compute and to send.

A greater benefit of being able to abort is that it gives us a chance to know for sure that the view does not reveal any unwanted information. The information we do not want to be revealed is ether our sk or which sk to the vk we have used. Having the ability to make sure this information does not leak makes aborting a great tool for securing ZK and WI for our ID-scheme.

3.2 What is a Signature?

Definition 3.8 *A* ***signature scheme*** *is made up of three polynomial-time algorithm (G, S, V), the key generator algorithm G, the signature algorithm S and the verification algorithm V. These algorithm are such that for every par (sk, vk) generated by G and a n-bit message μ, the following probability*

$$Pr[V(vk, \mu, S(sk, \mu)) = 1] = 1$$

holds.

Our three algorithms in a signature scheme are key generator algorithm, signature algorithm and verification algorithm. The key generator G produces two keys, namely the signature key, sk, and the verification key, vk. These keys then gets distributed to Peggy and Victor respectively. Peggy and Victor are then ready to run the signature and the verification algorithms to sign and verify, respectively.

Peggy with her sk, use the signature algorithm, S, to sign her message μ. Having her signature she can then send it to Victor for him to verify. Victor uses the verification algorithm, V, on vk from G and the signature from Peggy, and accepts or reject, depending on the outcome.

We know that the completeness in our signature follows from the ID-scheme that we built our signature on. Therefore we know that when both Peggy and Victor are honest the signature will succeed. In general when we use a signature scheme that is built upon an ID-scheme, the security from the ID-scheme follows over to the signature. Therefor we can trust that our signature has soundness and zero-knowledge. However, we now get some new problems. What if there is a third party that wants to forge a signature on a message?

3.2.1 Unforgeable

Definition 3.9 *A signature scheme (G, S, V) is* ***unforgeable*** *if for every polynomial-time forger F, after seeing the public key and the signatures for j messages of its choosing, $\{(\mu_1, S(sk, \mu_1)), ..., (\mu_j, S(sk, \mu_j))\}$, the probability that F can produce a signature $\mu, S(sk, \mu)$, such that $\mu \neq \mu_i \ \forall i$ and $V(vk, \mu, S(sk, \mu) = 1$, is negligible.*

We say that a signature is unforgeable if no forger F can forge a signature on a new message. In addition we can get another layer of security by adding the notion of *strongly unforgable*. This means that in addition to not being able to forge a new message the forger should not be able to forge a signature to a message they already have seen the signature to.

3.2.2 Random Oracle Model

Often when working with signatures we want to use hash functions. To provide proof of security on such signatures we need to be able to simulate the hash functions in some way. In order to so we need some strong assumptions, Fiat, Shamir, Bellare and Rogaway [3, 13] suggests to use a random function to simulate hashes. This has led to the random oracle model or ROM for short. The model simulate hashes as oracles that when queried to puts out random answers. The oracles also keep track of old queries so it can answer the same if a query is repeated. If we have a hash in our signature scheme with no weaknesses a security proof the model will ensure that the signature is secure.

3.2.3 Forking Lemma

The oracle replay attack is an attack where the attacker replays the attack using a different oracle, obtaining two different signature to one message. This opens a opportunity to break underlying security properties for our schemes. We follow Pointcheval and Regev [29] when proving the following theorem. But first we need to state a well known lemma in order to prove our nest theorem.

Lemma 3.1 *If $W \subset X \times Y$, such that $Pr[W(x, y)] = \epsilon$, there exists an $\Omega \subset X$ such that*

- *$Pr[z \in \Omega] \geq \epsilon/2$*
- *whenever $a \in \Omega$, $Pr[W(a, y)] \geq \epsilon/2$*

Theorem 1 ***(Forking Lemma)*** *Let A be a probabilistic polynomial time turing machine which only get the verification key as its input. If A can, with non-negligible probability, find a valid signature $(\boldsymbol{z}, c)$ for a message μ, then A with another oracle can replay and, with non-negligible probability, find a new valid signature for μ such that $(\boldsymbol{z}, c) \neq (\boldsymbol{z}', c')$.*

Proof Having a probabilistic polynomial time turing machine A we assume A is a no message attacker with a random type ω. A will then run an attack by sending Q queries, $q_1, ..., q_Q$ with Q a polynomial, to a random oracle, RO_1. In return the random oracle replies with $\rho_1, ..., \rho_Q$ respectively. These responses correspond to our random oracle RO_1 and querying to a different, RO_i $i \neq 1$, would result in different responses. A can now, with non-negligible probability, make a random choose of $\omega, \rho_1, ..., \rho_Q$ and output a valid signature $(\mu, \mathbf{z}, c)$. There is only a negligible probability that μ does not get queried, because of the randomness of our RF, thus μ is on of the q_i's with non-negligible probability, lets call it q_β. Because β is in between 1 and $q(n)$, we know there exists a polynomial P such that the probability of success over $\omega, \rho_1, ..., \rho_Q$ with q_β is $1/P(n)$.

We can use Lemma 3.2.3 together with our β and get a subset Ω_β of "good" ω's. An ω from this subset gives us a probability of success greater than $1/2P(n)$ $\omega, \rho_1, ..., \rho_Q$ with q_β. We use Lemma 3.2.3 again with β and ω to get a subset $\mathcal{Q}_{\beta,\omega}$

of “good” $(\rho_1, ..., \rho_{\beta-1})$’s. Now the attacker have probability $1/4P(n)$ of success with an attack over $\rho_\beta, ..., \rho_Q$. Having now $\beta, \omega, \rho_1, ..., \rho_{\beta-1}$ we use two random oracles, RO_1 and RO_2, to obtain $\rho_\beta, ..., \rho_Q$ and $\rho'_\beta, ..., \rho'_Q$ and get a non-negligible probability of finding two signatures $(\mu, \mathbf{z}, c)$ and $(\mu, \mathbf{z}', c')$ with $(\mathbf{z}, c) \neq (\mathbf{z}', c')$.
□

We have now proven that if we can, with non-negligible probability, find a valid signature we can find an other valid signature, also with non-negligible probability. Forking lemma can be a problem, but we have already used it when discussing ID-schemes and hence also already discussed ways to work against it. As long as we make sure the underlying ID-scheme to our signatures are WI and ZK we are safe from this lemma.

3.2.4 Sign with Abort

As with ID-schemes we want to include the notion of aborting in our signatures. This time it is a lot easier as we do not have to ask for a new challenge each time we abort. Now we merely need to compute a new challenge ourselves and Victor do not need to know how many times we tried to sign before we arrived at a suitable signature. Likewise, to the ID-schemer, the reason for wanting to abort is to have more control on the signatures we send out. When aborting we have the opportunity to make sure that our signature does not reveal any unwanted information, thus helping with WI and ZK requirement.

Simultaneously aborting make sure that the size of our signature is small. Whereas the effects of having a short signature is lost in an ID-scheme if we have to abort, this is not the case with signature schemes. Because we only need to run the protocol again, our self aborting has very little effect on the efficiency of the scheme, but having a short signature is certainly beneficial.

4 Going from ID-Schemes to Signatures

In this section we will present three different signature schemes, namley Fiat-Shamir, Luybashevsky and Dilithium. Fiat-Shamir is not a quantum safe crypto system, but it presents a very interesting method for transforming ID-schemes into signature schemes. This method is commonly known as the Fiat-Shamir method, and is used in both Luybashevsky and Dilithium, who are both quantum safe. We will therefore start by presenting Fiat-Shamir, and then move on to Luybashevsky and Dilithium.

4.1 Fiat-Shamir

Fiat-Shamir is a method for convert an interactive proof of knowledge, or ID-schemes as we know them, to a signature scheme. The method was created by Fiat and Shamir in 1986 [13], and uses hash functions to simulate a random challenge Victor creates in the ID-scheme. This way, Peggy does not need to have more than one interaction with Victor to prove herself to him. As we are mostly interested in the method and not the scheme in itself we will not bother to look closer at the security of this scheme.

4.1.1 ID-Scheme

To describe how the method works we first need an ID-scheme to work out from. The example we are using is the original from [13]. Below we present the two parts of an ID-scheme, namely key generation and interactive protocol.

The key generator algorithm produces a signature key by taking I and using it with our random function. The output from the function is then set as the verification key, and the square root of its inverses as the signature key. Both vk and sk consist of k random values from $[0, n)$, where n is the product of two primes.

When the keys are distributed between Peggy and Victor, they can run the interactive protocol by having Peggy picking t random values. She square each of them and send them all to Victor. Victor respond to Peggys commitment by sending a matrix back. Peggy can then compute her response using her key, commitment and the matrix from Victor and then send the final result to Victor. Lastly Victor uses his key, the commitment and his matrix to conclude wheather or not he trust Peggy.

Key generation:

```
(1)      for $i = 0$ to $k$
    (1.1)      set $v_i$ to $f(I, i)$
    (1.2)      set $s_i$ to be the square root of $v_i^{-1}$
(2)      set $vk = (v_1, ..., v_k)$, $sk = (s_1, ..., s_k)$
```

Interactive protocol:

```
(1)      for $j = 0$ to $t$
    (1.2)      Peggy pick $r_j$ at random from $[0, n)$
    (1.2)      Peggy sends $x_j = r_j^2$ to Victor
(2)      Victor sends a random binary $k \times t$ matrix $\mathbf{C}$ to Peggy
(3)      for $j = 0$ to $t$
    (3.1)      Peggy sends $y_j = r_j \prod_{\mathbf{C}_{ij}=1} s_i \bmod(n)$ to Victor
(4)      if $\forall i \in [0, t]$ $x_j = y_j^2 \prod_{\mathbf{C}_{ij}=1} v_i \bmod(n)$ Victor accept, else Victor reject
```

This ID-scheme has the structure of the sigma protocol, and we see that here $\alpha = (x_1, .., x_t)$, $\beta = c$ and $\gamma = (y_1, ..., y_t)$.

4.1.2 Signature Scheme

Fiat and Shamir uses a function, F, that is indistinguishable from a truly random function for Peggy to create a challenge for themselves. The scheme of Fiat and Shamir is illustrated in Figure 3 and consist of three algorithms. The key generation algorithm is as in the ID-scheme.

We separate the interactive protocol in to two algorithms in the signature. Here point 1-3 is contained in the signature algorithm, run by Peggy, and point 4 is set as the verification algorithm, run by Victor. In the signature algorithm all the previous points done by Peggy is as before. However, point 2 was previously done by Victor, but will now be computed by Peggy using a function to simulate the randomness and unpredictability from Victor. The function used has to be sufficiently hard to find inverses with or else the signature looses its security. This results in Peggy sending $\mu, I, \mathbf{C}, y$ as her signature.

The verification algorithm is now consists of point 4. It computes the challenge itself, and then compeers the result to the two matrices. If they are the same Victor accept Peggys signature, otherwise he rejects her.

To show that the signature has completeness we only need to show that
$F(\mu, x_1, ..., x_t) = F(\mu, z-1, ..., z_t)$, this is easy to see by the following equation.

$$z_i = y_i^2 \prod_{e_{ij}} v_j = r_i^2 \prod_{e_{ij}} s_j^2 v_j = x_i$$

```
Algorithm: Key Generator
  Input: $I$
1 for $j = 1, ..., k$ do
2     $v_j = f(I, j)$;
3     $s_j \leftarrow$ square root of $v_1^{-1}$ mod$(n)$;
4 Return $sk = (s_1, ..., s_k), vk = (v_1, ..., v_k)$
```

```
Algorithm: Signature
  Input: $\mu, sk$
1 for $i = 0, ..., t$ do
2     $r_i \leftarrow [0, n)$;
3     $x_i = r_i^2$ mod$(n)$;
4 $F(\mu, x_1, ..., x_t) \rightarrow$ string of bits $\beta$;
5 $\mathbf{C} \leftarrow k \times t$-matrix of the first $kt$ bits from $\beta$;
6 for $i = 0, ..., t$ do
7     $y_i = r_i \prod_{\mathbf{C}_{ij}=1} s_j$ mod$(n)$;
8 Return $(\mu, \mathbf{C}, y)$
```

```
Algorithm: Verification
  Input: $\mu, vk, \mathbf{C}, y$
1 for $j = 1, ..., k$ do
2     $v_j = f(I, j)$;
3 for $i = 0, ..., t$ do
4     $z_i = y_i^2 \prod_{\mathbf{C}_{ij}=1} v_j$ mod$(n)$;
5 $F(\mu, z_1, ..., z_t) \rightarrow$ string of bits $\beta'$;
6 $\mathbf{C}' \leftarrow k \times t$-matrix of the first $kt$ bits from $\beta'$;
7 if $\mathbf{C} = \mathbf{C}'$ then
8     Accept
9 else
10    Reject
```

Figure 3: Fiat-Shamir

4.2 Lyubashevsky

Lyubashevsky [25] presents a signature scheme using the Fiat-Shamir method for going from an ID-scheme to a signature using a function to simulate Victors randomness to create a random challenge. We go from an ID-scheme presented below to the signature scheme presented in Figure 4. We will also use a hash function from our hash function family $H(R, D, m)$ that will provide the security in this scheme, also agings quantum computers.

4.2.1 ID-Scheme

The key generation is done by setting the signature key to be a random value and the hash from $H(R, D, m)$. Then the verification key is made to fit the signature key by setting it to be the hash of the random value, using the same hash, and the hash itself.

We have four stages in the interactive protocol. In the first, Peggy chooses m random values and hashes them before sending them to Victor as her commitment. Victor simultaneously chooses a random value and sends them as his challenge. Peggy then computes her response in point 3, if it is within her preferred range she sends her signature over to Victor. Otherwise abort and she starts the protocol from point 1 again. Lastly Victor either accept or reject Peggy based on whether the equation in point 4 is correct or not.

Key generation.
(1) $sk = (\mathbf{s} \leftarrow D_s^m,\ h \leftarrow H(R, D, m))$
(2) $vk = (h(\mathbf{s}),\ h \leftarrow H(R, D, m))$

Interactive protocol:
(1) $\mathbf{y} \leftarrow D_y^m$, Peggy set $Y = h(\mathbf{y})$ and sends Y to Victor
(2) $c \leftarrow D_c$, Victor sends c to Peggy
(3) $\mathbf{z} = \mathbf{s}c + \mathbf{y}$
 (3.1) if $\mathbf{z} \in G^m$ Peggy send $\mathbf{z}$ to Victor
 (3.2) else Peggy aborts and runs the protocol again.
(4) Victor accepts if $h(\mathbf{z}) = Sc + Y$ and $\mathbf{z} \in G^m$, else he rejects

Parameter	Definition	Sample Instantiations			
n	integer that is a power of 2	512	512	512	1024
m	any integer	4	5	8	8
σ	any integer	127	2047	2047	2047
κ	integer s.t. $2^\kappa \binom{n}{\kappa} \geq 2^{160}$	24	24	24	21
p	integer $\approx (2\sigma+1)^m 2^{-\frac{128}{n}}$	$2^{31.7}$	$2^{59.8}$	$2^{95.8}$	$2^{95.9}$
R	$\mathbb{Z}_p[x]/\langle x^n+1\rangle$				
D	$\{g \in R \mid \|\|g\|\|_\infty \leq nm\sigma\kappa\}$				
D_s	$\{g \in R \mid \|\|g\|\|_\infty \leq \sigma\}$				
D_c	$\{g \in R \mid \|\|g\|\|_1 \leq \kappa\}$				
D_y	$\{g \in R \mid \|\|g\|\|_\infty \leq nm\sigma\kappa\}$				
G	$\{g \in R \mid \|\|g\|\|_\infty \leq nm\sigma\kappa - \sigma\kappa\}$				

Table 2: Table of Parameters for Lyubashevskys.

4.2.2 Signature Scheme

In the signature the key generation is the same as the in ID-scheme. The interactive protocol, however, gets split up into the signature and verification algorithm. Here, like with Fiat-Shamir, point 1-3 in the interactive protocol is put into the signature algorithm and point 4 in the verification algorithm.

Point 1 and 3 are the same in the algorithm as in the protocol. Point 2 is changed as it is no longer Victor that constructs the challenge, but Peggy doing it for herself. This is done by using the Fiat-Shamir method and having Peggy using a random oracle, RO. The random oracle takes the hash of commitment and the message as input, and outputs randomness to simulate the challenge we normally would get from Victor. Peggy is then able to use this randomness as her challenge, computes her signature, and sends $\mu, \mathbf{z}, c$ to Victor.

The verification algorithm is set to be point 4 from the protocol with some adjustments. We now want the challenge Peggy computed to be the same as the output from the random oracle when setting $h(\mathbf{z}) - Sc$ and the message as its input. If this is correct, Victor accepts, otherwise he rejects.

It is easy to see that if we have an honest Peggy she will be accepted because of the linearity of our hash functions, as shown below.

$$h(\mathbf{z}) - Sc = h(\mathbf{s}c + \mathbf{y}) - h(\mathbf{s})c = h(\mathbf{s})c + h(\mathbf{y}) - h(\mathbf{s})c = h(\mathbf{y})$$

Algorithm: Key Generator

```
1 $sk : h \leftarrow H(R, D, m),\ \mathbf{s} \leftarrow D_s^m$;
2 $vk : h \leftarrow H(R, D, m),\ S \leftarrow h(\mathbf{s})$;
```

Algorithm: Signature

```
  Input: $\mu, sk$
1 $\mathbf{y} \leftarrow D_y^m$;
2 $c \leftarrow RO(h(\mathbf{y}), \mu)$;
3 $\mathbf{z} \leftarrow \mathbf{s}c + \mathbf{y}$;
4 if $\mathbf{z} \notin G^m$ then
5     run again;
6 Return $(\mu, \mathbf{z}, c)$
```

Algorithm: Verification

```
  Input: $\mu, \mathbf{z}, c, vk$
1 if $\mathbf{z} \in G^m$ and $c = RO(h(\mathbf{z}) - Sc, \mu)$ then
2     Accept;
3 else
4     Reject;
```

Figure 4: Lyubashevskys

Similarly to the way that completeness follows from the ID-schemes, the signatures ZK security follows from the ID-scheme. The protocol may need to run a few times to make sure $\mathbf{z} \in G^m$, but this is acceptable, and Lyubashevsky [25] state that we will have a usable $\mathbf{z}$ in under 3 attempts, with the parameters given in Table 2. This is very acceptable.

4.2.3 Security

Now we prove that the probability that a random forger can forge a signature for some message is negligible. This should hold, whether or not, the forger has seen a signature for the message before. The following theorem shows that this signature scheme is strongly unforgeable.

Theorem 2 *If the signature scheme in Figure 4 with the parameters in Table 2 is not strongly unforgeable, then there is a polynomial-time algorithm that can solve $SVP_\gamma(\Lambda(I))$ for $\gamma = \tilde{O}(n^2)$ for every lattice $\Lambda(I)$ corresponding to an ideal in the ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$.*

We will draft the proof: if we do not have strong unforgeability there exists a forger F that can forge a signature to some message. If this is the case, we have an F that is able to, without knowing sk, get a valid $\mathbf{z}$ for some random c and μ. In other words, F is able to find

$$\begin{gathered} RO(h(\mathbf{z}) - Sc) = RO(h(\mathbf{z}') - Sc') \\ \Rightarrow \\ h(\mathbf{z}) - Sc = h(\mathbf{z}') - Sc' \\ \Rightarrow \\ h(\mathbf{z} - \mathbf{z}') = S(c - c') \end{gathered}$$

This means that F has access to a way of finding a collision in our hash function h. We have already shown, in Proposition 2.1 and 2.2, that having a collision in our hash function from our hash function family give us a way to solve SVP$_\gamma(\Lambda(I))$, where I is an ideal in R_p.

The security for the scheme then follows from the fact that we believe it is very hard to solve $\mathrm{SVP}_\gamma(\Lambda(I))$, especially when working with ideal lattices. It thus follows from our theorem that this signature scheme is strongly unforgeble.

□

4.3 Dilithium

As with Lyubasheavskys scheme, our new scheme Dilithium is also based on the Fiat-Sahmir method. We can thus find similarities between the two schemes. Dilithium is one of the contestants for NISTs call for post-quantum cryptographic standards and created by Ducas et al. [11].

4.3.1 ID-Scheme

Before we look at the actual signature scheme we want to know the underlying ID-scheme, just as we did with Fiat-Shamir and Luybashevsky. This ID-scheme is, like the other ID-schemes, a sigma protocol.

The key generator algorithm starts by choosing some random values to use in computing the keys. There are three functions defined outside of the scheme used to compute the parameters for the keys: ExpandA and CRH. Both are based on hash functions, which we will explain in more detail in the next section when discussing why they help upgrade the scheme. Power2round is an easy function described in Figure 6. After using all the functions to further randomise and construct suitable variables we end up with the verification key $\rho, \mathbf{t}_1$ and the signature key $\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_2$.

The interactive protocol starts by having both Peggy and Victor compute $\mathbf{A}$ using ExpandA before it continues on as a normal sigma protocol. Peggy then masks over her message with that hash based function CRH. Using another hash based function, she constructs a random value and feeds $\mathbf{Ay}$ to the function HighBits, described in Figure 6, getting her commitment w_1, before sending it off. Victor chooses a random challenge and sends it to Peggy. The final response from Peggy now consists of two elements $\mathbf{z}$ and $\mathbf{h}$, where $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ and $\mathbf{h}$ is computed using the MakeHint function. Peggy checks if these are up to her standards before sending them to Victor. If the elements are not within her preferred range she aborts and starts over again. Lastly, Victor uses the functions CRH and Use-Hint to compute the values necessary to verify Peggy and either accepts or rejects.

Key generation:
(1) $\rho \leftarrow \{0,1\}^{256}$
(2) $K \leftarrow \{0,1\}^{526}$
(3) $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow D_\mu^l \times D_\mu^k$
(4) $\mathbf{A} \in R_q^{k \times l} = \text{ExpandA}(\rho)$
(5) $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
(6) $(\mathbf{t}_1, \mathbf{t}_2) = \text{Power2round}(\mathbf{t}, d)$
(7) $tr \in \{0,1\}^{384} = \text{CRH}(\rho||\mathbf{t}_1)$
(8) $pk = (\rho, \mathbf{t}_1)$ and $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_2)$

Interactive protocol:
(1) $\mathbf{A} \in R_q^{k \times l} = \text{ExpandA}(\rho)$ is computed by both Peggy and Victor
(2) Peggy set $\kappa = 0$ and compute:
  (2.1) $m \in \{0,1\}^{384} = \text{CRH}(tr||\mu)$
  (2.2) $\mathbf{y} \in D_{\gamma_1 - 1}^t = \text{ExpandMask}(K||m||\kappa)$
  (2.3) $\mathbf{w} = \mathbf{A}\mathbf{y}$
  (2.4) $\mathbf{w}_1 = \text{Highbits}(\mathbf{w}, 2_{\gamma_2})$ and sends w_1 as her commitment
(3) Victor chooses c at random as his challenge and sends it to Peggy
(4) Peggy sets $(\mathbf{z}, \mathbf{h}) = \perp$ and compute:
  (4.1) $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$
  (4.2) $(\mathbf{r}_1, \mathbf{r}_2) = \text{Decompose}(\mathbf{w} - c\mathbf{s}_2, 2_{\gamma_2})$
  (4.3) If $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ or $||\mathbf{r}_2||_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1$ Peggy aborts and starts again from (2) with $\kappa = \kappa + 1$
  (4.4) $\mathbf{h} = \text{MakeHint}(-c\mathbf{t}_2, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_2, 2_{\gamma - 2})$
  (4.5) If $||c\mathbf{t}_2||_\infty \geq \gamma_2$ or the number of 1's in $\mathbf{h}$ is greater than ω Peggy aborts and starts again from (2) with $\kappa = \kappa + 1$
  (4.6) Peggy sends $(\mathbf{z}, \mathbf{h})$ to Victor
(5) Victor sets $m \in \{0,1\}^{384} = \text{CRH}(\text{CRH}(\rho||\mathbf{t}_1)||\mu)$ and
$\mathbf{w}_1' = \text{UseHint}(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2_{\gamma_2})$
  (5.1) If $||\mathbf{z}||_\infty < \gamma_1 - \beta$ and $c = \text{H}(\mu||\mathbf{w}_1')$ and number of 1's in $\mathbf{h}$ is less than ω, Victor accepts Peggy, otherwise he rejects

4.3.2 Signature Scheme

The three algorithms that make up our scheme are illustrated in Figure 5 with some additional functions used in the scheme illustrated in Figure 6. First of is the key generation algorithm which is done the same way as in the key generation for our ID-scheme. The two keys $pk = (\rho, \mathbf{t}_1)$ and $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_2)$ are then distributed to Peggy and Victor respectively.

Next we split the interactive protocol into the signature algorithm and verification algorithm. The signature algorithm consists of point 1-4. All points except 3 is done in the same way as in the interactive protocol. In point 3, however, Peggy uses a hash often referred to as an “inside-out” version of the Fisher-Yates shuffle. This hash takes in Peggys already hash message and her commitment and hashes these to a random element on the ball B_{60}. Peggy is able to complete her signature as in the interactive protocol with this variable and sends off $\mathbf{z}, \mathbf{h}$ and c as her signature to Victor.

The verification algorithm now consists of point 1 and 5 from the interactive protocol. As in the interactive protocol we need Victor to have accses to the matrix $\mathbf{A}$, therefore we have him compute it for himself. Having this, Victor then accepts or rejects Peggy on the same bases as in the interactive protocol.

Algorithm: Key Generator

```
1  $\rho \leftarrow \{0,1\}^{256}$;
2  $K \leftarrow \{0,1\}^{526}$;
3  $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow D_\mu^l \times D_\mu^k$;
4  $\mathbf{A} \in R_q^{k \times l} = \text{ExpandA}(\rho)$;
5  $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$;
6  $(\mathbf{t}_1, \mathbf{t}_2) = \text{Power2round}(\mathbf{t}, d)$;
7  $tr \in \{0,1\}^{384} = \text{CRH}(\rho || \mathbf{t}_1)$;
8  **Return** $(pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_2))$
```

Algorithm: Signature

```
   **Input:** $(sk, \mu)$
1  $\mathbf{A} \in R_q^{k \times l} = \text{ExpandA}(\rho)$;
2  $m \in \{0,1\}^{384} = \text{CRH}(tr || \mu)$;
3  $\kappa = 0$;
4  $(\mathbf{z}, \mathbf{h}) = \perp$;
5  **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
6      $\mathbf{y} \in D_{\gamma_1 - 1}^t = \text{ExpandMask}(K || m || \kappa)$;
7      $\mathbf{w} = \mathbf{A}\mathbf{y}$;
8      $\mathbf{w}_1 = \text{HighBits}(\mathbf{w}, 2_{\gamma_2})$;
9      $c = \text{H}(m || \mathbf{w}_1)$;
10     $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$;
11     $(\mathbf{r}_1, \mathbf{r}_2) = \text{Decompose}(\mathbf{w} - c\mathbf{s}_2, 2_{\gamma_2})$;
12     **if** $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ *or* $||\mathbf{r}_2||_\infty \geq \gamma_2 - \beta$ *or* $\mathbf{r}_1 \neq \mathbf{w}_1$ **then**
13         $(\mathbf{z}, \mathbf{h}) = \perp$;
14         $\kappa = \kappa + 1$;
15     **else**
16         $\mathbf{h} = \text{MakeHint}(-c\mathbf{t}_2, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_2, 2_{\gamma - 2})$;
17         **if** $||c\boldsymbol{t}_2||_\infty \geq \gamma_2$ *or the number of 1's in* $\boldsymbol{h}$ *is greater than* $\omega$ **then**
18             $(\mathbf{z}, \mathbf{h}) = \perp$;
19             $\kappa = \kappa + 1$;
20 **Return** $\sigma = (\mathbf{z}, \mathbf{h}, c)$
```

Algorithm: Verification

```
   **Input:** $(pk, \mu, \sigma)$
1  $\mathbf{A} \in R_q^{k \times l} = \text{ExpandA}(\rho)$;
2  $m \in \{0,1\}^{384} = \text{CRH}(\text{CRH}(\rho || \mathbf{t}_1) || \mu)$;
3  $\mathbf{w}'_1 = \text{UseHint}(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2_{\gamma_2})$;
4  **if** $||\mathbf{z}||_\infty < \gamma_1 - \beta$ *and* $c = H(\mu || \mathbf{w}'_1)$ *and number of 1's in* $\mathbf{h}$ *is less than*
   $\omega$ **then**
5      **Return** Accept
6  **else**
7      **Return** Reject
```

Figure 5: Dilithium

```
Algorithm: Power2round
  Input: $r, d$
  $r = r \bmod(q)$;
  $r_1 = r \bmod(2^d)$;
  $r_2 = (r - r_1)/2^d$;
  Return $(r_1, r_2)$
```

```
Algorithm: Decompose
  Input: $r, \alpha$
  $r = r \bmod(q)$;
  $r_1 = r \bmod(\alpha)$;
  if $r - r_1 = q - 1$ then
      $r_2 = 0$;
      $r_1 = r_1 - 1$;
  else
      $r_2 = (r - r_1)/\alpha$;
  Return $(r_1, r_2)$
```

```
Algorithm: HighBits
  Input: $r, \alpha$
  $(r_1, r_2) = \text{Decompose}(r, \alpha)$;
  Return $r_2$
```

```
Algorithm: LowBits
  Input: $r, \alpha$
  $(r_1, r_2) = \text{Decompose}(r, \alpha)$;
  Return $r_1$
```

```
Algorithm: MakeHint
  Input: $z, r, \alpha$
  $r_1 = \text{HighBits}(r, \alpha)$;
  $v_1 = \text{HighBits}(r + z, \alpha)$;
  if $r_1 \neq v_1$ then
      Return True
  else
      Return False
```

```
Algorithm: UseHint
  Input: $h, r, \alpha$
  $m = (q - 1)/\alpha$;
  $(r_1, r_2) = \text{Decompose}(r, \alpha)$;
  if $h = 1$ and $r_1 > 0$ then
      Return $(r_2 + 1) \bmod(m)$
  else if $h = 1$ and $r_1 \leq 0$ then
      Return $(r_2 - 1) \bmod(m)$
  else
      Return $r_2$
```

Figure 6: Helping Algorithms

These helping functions are created to extract "high order" and "low order" bits, to make us able to recover high order bits without having to store them. This is done by making a 1-bit hint to help us compute the high order bits, and thus recover them. The following two lemmas give some properties to these helping function which are crucial for the security and completeness of the signature. We will therefore proceed to prove them.

Lemma 4.1 *Suppose q and α are positive integers satisfying $q > 2\alpha$, $q \equiv 1$ mod(α) and α even. Let $\boldsymbol{r}$ and $\boldsymbol{z}$ be vectors of elements in R_q where $||\boldsymbol{z}||_\infty \leq \alpha/2$, and let $\boldsymbol{h}, \boldsymbol{h}'$ be vectors of bits. Then the HighBits, MakeHint and UseHint algorithm satisfy the following properties:*

1. *UseHint(MakeHint($\boldsymbol{z}, \boldsymbol{r}, \alpha$), $\boldsymbol{r}, \alpha$)=HighBits($\boldsymbol{r} + \boldsymbol{z}, \alpha$)*
2. *Let $\boldsymbol{v}$=UseHint($\boldsymbol{h}, \boldsymbol{r}, \alpha$), then $||\boldsymbol{r} - \boldsymbol{v} \cdot \alpha||_\infty \leq \alpha + 1$. If the number of 1's in $\boldsymbol{h}$ is ω, then all except at most ω coefficients of $\boldsymbol{r} - \boldsymbol{v} \cdot \alpha$ will have magnitude at most $\alpha/2$ after centered reduction modulo q.*
3. *If UseHint($\boldsymbol{h}, \boldsymbol{r}, \alpha$) =UseHint($\boldsymbol{h}', \boldsymbol{r}, \alpha$), for any $\boldsymbol{h}, \boldsymbol{h}'$, then $\boldsymbol{h} = \boldsymbol{h}'$.*

Proof(1) Decompose($\mathbf{r}, \alpha$ output two integer r_1, r_2, with $||r_1||_\infty \leq \alpha/2$ and $0 < r_2 < (q-1)/\alpha$. If we put v=HighBits($\mathbf{r} + \mathbf{z}, \alpha$) then either $v = r_2$, $v = r_2 + 1$ or $v = r_2 - 1$, since $||\mathbf{z}||_\infty \leq \alpha/2$. If $v = r_2$ MakeHint outputs 0 and if $v \neq r_2$ MakeHint outputs 1. UseHint then outputs $r_2, r_2 + 1$ or $r_2 - 1$ depending on the hint if gets from MakeHint. If UseHint gets 0 it outputs r_2, if it receives 1 it outputs $r_2 \pm 1$ depending on whether or not r_1 is greater than 0. We then see that UseHint(MakeHint($\mathbf{z}, \mathbf{r}, \alpha$), $\mathbf{r}, \alpha$)=HighBits($\mathbf{r} + \mathbf{z}, \alpha$).

Proof(2) UseHint produces three cases and we will go through than with $(r_1, r_2) =$ Decompose($\mathbf{r}, \alpha$).

1. ($h = 0$): We have $v = r_2$ then

$$r - v \cdot \alpha = r_2 \cdot \alpha + r_1 - r_2 \cdot \alpha = r_1$$

Which by definition hash absolute value $\alpha/2$.

2. ($h = 1$ and $r_1 > 0$): We have $v = r_2 + 1 - \kappa \cdot (q-1)/\alpha$ with $\kappa = 0$ or $\kappa = 1$, then

$$\begin{aligned} r - v \cdot \alpha &= r_2 \cdot \alpha + r_1 - (r_2 + 1 - \kappa \cdot (q-1)/\alpha) \cdot \alpha \\ &= -\alpha + r_1 + \kappa \cdot (q-1) \end{aligned}$$

and the latter has magnitude $\leq \alpha$.

3. ($h = 0$ and $r_1 \leq 0$): Now we have $v = r_2 - 1 + \kappa \cdot (q-1)/\alpha$ with $\kappa = 0$ or $\kappa = 1$, then

$$\begin{aligned} r - v \cdot \alpha &= r_2 \cdot \alpha + r_1 - (r_2 - 1 + \kappa \cdot (q-1)/\alpha) \cdot \alpha \\ &= \alpha + r_1 - \kappa \cdot (q-1) \end{aligned}$$

   which has magnitude $\leq \alpha + 1$.

Proof(3) Start by setting $\mathbf{h} \neq \mathbf{h}'$, and we get UseHint$(0, \mathbf{r}, \alpha) = r_2$ and UseHint$(1, \mathbf{r}, \alpha) = r_2 \pm 1 \bmod((q-1)/\alpha)$. We know that $(q-1)/\alpha \geq 2$ and thus $r_2 \neq r_2 \pm 1 \bmod((q-1)/\alpha)$.

$\square$

Lemma 4.2 *If $||\boldsymbol{s}||_\infty \leq \beta$ and $||LowBits(\boldsymbol{r}, \alpha)||_\infty < \alpha/2 - \beta$, then*

$$HighBits(\boldsymbol{r}, \alpha) = HighBits(\boldsymbol{r} + \boldsymbol{s}.\alpha).$$

Proof $||\text{LowBits}(\mathbf{r}, \alpha)||_\infty < \alpha/2 - \beta$ we have that $\mathbf{r} = r_2 \cdot \alpha + r_1$ with $-\alpha/2 + \beta < r_1 \leq \alpha/2 + \beta$. Then we get that $\mathbf{r} + \mathbf{s} = r_2 \cdot \alpha + (r_1 + \mathbf{s})$ with $-\alpha/2 < r_1 + \mathbf{s} \leq \alpha/2$ which gives us that $\mathbf{r} + \mathbf{s} = r_1\mathbf{s} \bmod(\alpha)$ and thus

$$(\mathbf{r} + \mathbf{s}) - (\mathbf{s} + \mathbf{s} \bmod(\alpha)) = r_2 \cdot \alpha = \mathbf{r} - (\mathbf{r} \bmod(\alpha))$$

Which completes the proof.

$\square$

4.3.3 Security

It is worth noting that there are a lot of set parameters in our algorithms. Ducas et al. [11] present some suggested input for these parameters with different levels of security. We present these suggested parameters in Table 3.

Parameter	Sample Instantiations			
	weak	medium	recommended	very high
q	8380417	8380417	8380417	8380417
d	14	14	14	14
wight of c	60	60	60	60
$\gamma_1 = (q-1)/16$	523776	523776	523776	523776
$\gamma_2 = \gamma_1/2$	261888	261888	261888	261888
k	3	4	5	6
l	2	3	4	5
η	7	6	5	3
β	375	325	275	175
ω	64	80	96	120

Table 3: Table of Parameters for Dilithium.

To prove the security for our signature we first need to prove a proposition and two lemmas, all following the structures of Theorem 3.2, Lemma 4.9 and Lemma 4.10 in [21]. In order to present the proposition we need to simplify our signature. The simplification only strips the scheme down to focus on the security, and the security from the simplified signature easily follows over to Dilithium. The signing algorithm and verification algorithm will be as in Figure 7.

Proposition 4.1 *Given an ID-scheme with ϵ_{zk}-perfect naHVZK and α bits of min entropy. For a SUF-CMA quantum adversary A issuing at most Q_H queries to the quantom random oracle QRO and Q_S to the signature oracle Sing, there exists a UF-NMA quantum adversary B and a CUR quantum adveraty C, where A, B and C are all in polynomial time, such that*

$$Adv_{SIG}^{SUF-CMA_1}(A) \leq Adv_{SIG}^{UF-NMA}(B) + 2^{-\alpha+1} + Adv_{ID}^{CUR}(C) + \kappa_m Q_S \cdot \epsilon_{zk}$$

Proof To prove this we need to look at three games which are described in Figure 8, 9 and 10. In the first game, lets call it $Game_1$ described in Figure 8, the signing oracle uses a perfect random function to derive its randomness. Our adversary does not have access to these functions. Only one signature query is allowed in this game and thus we get that

$$\mathbf{Pr}[Game_1^A \Rightarrow 1] = Adv_{SIG}^{SUF-CMA_1}(A)$$

```
Algorithm: Signature
  Input: $sk, \mu$
1 $\kappa = 0$;
2 while $Z = \perp$ and $\kappa \leq \kappa_m$ do
3     $\kappa = \kappa + 1$;
4     $(W, St) \leftarrow P_1(sk)$;
5     $c = H(W||\mu)$;
6     $Z \leftarrow P_2(sk, W, c, St)$;
7 if $Z = \perp$ then
8     Return $\sigma = \perp$
9 Return $\sigma' = (c, Z)$
```

```
Algorithm: Verification
  Input: $pk, \mu, \sigma'$
1 $\sigma = (W, Z) \in \text{WSet} \times \text{ZSet}$;
2 $c = H(W||\mu)$;
3 Return $V(pk, W, c, Z) \in \{0, 1\}$
```

Figure 7: Simplified Signature (SIG)

The second game, $Game_2$ illustrated in Figure 9, uses a naHVZK signature algorithm to sign M and the quantum random oracle, QRO, is patched accordingly. When querying query M we want to do this with an integer κ_M between 1 and κ_m such that $(W, c, Z) = Sim(pk; RF(M||\kappa))$ and $Z \neq \perp$. If this is not possible we set $\kappa_M = \perp$, and end up with the following computations

$$(W_M, c_M, Z_M) = \text{GetTrans}(M) = \begin{cases} \text{Sim}(pk; RF(M||\kappa_M)) & 1 \leq \kappa_M \leq \kappa_m \\ (\perp, \perp, \perp) & \kappa_M = \perp \end{cases}$$

The signature for M is then

$$\sigma_M = (W_M, Z_M)$$

We want to be sure that σ_M is a valid signature on M, this is done by patching our QRO such that $QRO(W||M) = c_m$ if and only if $W = W_M$. The properties

```
Algorithm: $Game_1$
1 $(pk, sk) \leftarrow \mathrm{IGen}(\mathrm{par};$
2 $(\mu^*, \sigma^*) \leftarrow A^{|H\rangle, \mathrm{Sign}_1}(pk);$
3 $\sigma^* = (W^*, Z^*);$
4 $c^* = \mathrm{RO}(W^* || \mu^*);$
5 if $\mu^* \notin \mathcal{M}$ and $V(pk, W^*, c^*, z^*)$ then
6     Return accept
7 else
8     Return reject
```

```
Algorithm: $\mathrm{Sign}_1$
  Input: $\mu$
1 if $\mu \in \mathcal{M}$ then
2     Return $\perp$
3 $\mathcal{M} = \mathcal{M} \cup \mu;$
4 $(W_\mu, c_\mu, z_\mu) = \mathrm{GetTrans}(\mu);$
5 Return $\sigma_\mu = (W\mu, Z_\mu)$
```

```
Algorithm: GetTrans
  Input: $\mu$
1 $\kappa = 0;$
2 while $Z_\mu = \perp$ and $\kappa \leq \kappa_m$ do
3     $\kappa = \kappa + 1;$
4     $(W_\mu, St) = \mathrm{P}_1(sk; \mathrm{RF}(0||\mu||\kappa));$
5     $c_\mu = \mathrm{RO}(W_\mu||\mu);$
6     $Z_\mu = \mathrm{P}_2(sk, W_\mu, c_\mu, St; \mathrm{RF}(1||\mu||\kappa));$
7 if $Z_\mu = \perp$ then
8     $(W_\mu, c_\mu, Z_\mu) = (\perp, \perp, \perp)$
9 Return $(W_\mu, c_\mu, Z_\mu)$
```

Figure 8: Game_1

```
Algorithm: $Game_2$
1 $(pk, sk) \leftarrow \mathrm{IGen}(\mathrm{par};$
2 $(\mu^*, \sigma^*) \leftarrow A^{|H\rangle, \mathrm{Sign}_1}(pk);$
3 $\sigma^* = (W^*, Z^*);$
4 $c^* = \mathrm{QRO}(W^*||\mu^*);$
5 if $\mu^* \notin \mathcal{M}$ and $V(pk, W^*, c^*, z^*)$ then
6     Return accept
7 else
8     Return reject
```

```
Algorithm: $\mathrm{Sign}_1$
  Input: $\mu$
1 if $\mu \in \mathcal{M}$ then
2     Return $\perp$
3 $\mathcal{M} = \mathcal{M} \cup \mu;$
4 $(W_\mu, c_\mu, z_\mu) = \mathrm{GetTrans}(\mu);$
5 Return $\sigma_\mu = (W\mu, Z_\mu)$
```

```
Algorithm: QRO
  Input: $W, \mu$
1 $(W_\mu, c_\mu, Z_\mu) = \mathrm{GetTrans}(\mu);$
2 if $W = W_\mu$ then
3     Return $c = c_\mu$
4 else
5     Return $c = \mathrm{RO}(w||\mu)$
```

```
Algorithm: GetTrans
  Input: $\mu$
1 $\kappa = 0;$
2 while $Z_\mu = \perp$ and $\kappa \leq \kappa_m$ do
3     $\kappa = \kappa + 1;$
4     $(W_\mu, c_\mu, Z_\mu) = \mathrm{Sim}(pk; \mathrm{RF}(\mu||\kappa));$
5 if $Z_\mu = \perp$ then
6     $(W_\mu, c_\mu, Z_\mu) = (\perp, \perp, \perp)$
7 Retunr $(W_\mu, c_\mu, Z_\mu)$
```

Figure 9: Game_2

from naHVZK also gives us that the statistical distance from σ_M given our simulation is at most $\kappa_m \epsilon_{zk}$ from a signature given by $Game_1$. This together gives us the observation

$$|\Pr[Game_2^A \Rightarrow 1] - \Pr[Game_1^A \Rightarrow 1]| \leq \kappa_m Q_S \cdot \epsilon_{zk}$$

Lastly we have $Game_3$, illustrated in Figure 10, which only differs from $Game_2$ if $W_{M^*} = W^*$, $(M^*, \sigma^*) \notin \mathcal{M}$ and $V(pk, W^*, c^*, Z^*) = 1$. We now have two cases we need to consider, namely $(M^*, \cdot) \notin \mathcal{M}$ and $(M^*, \cdot) \in \mathcal{M}$. In the first case, our adversary has not yet queried a signature and it has α bits of min-entropy, this gives us $\Pr[W_{M^*} = W^*]$ and thus

$$|\Pr[Game_3^A \Rightarrow 1] - \Pr[Game_2^A \Rightarrow 1]| \leq 2^{-\alpha+1}$$

The other case is that our adversary has obtained a signature σ_{M^*} on a message M^* and submits a correct forgery $\sigma^* = (W^*, Z^*)$ that satisfy $W^* = W_{M^*}$ and $Z^* \neq Z_{M^*}$. The probability for this is bounded by the advantage of a CUR adversary C. Hence we have that

$$|\Pr[Game_3^A \Rightarrow 1] - \Pr[Game_2^A \Rightarrow 1]| \leq Adv_{ID}^{CUR}(C)$$

Combining these two cases gives us

$$|\Pr[Game_3^A \Rightarrow 1] - \Pr[Game_2^A \Rightarrow 1]| \leq Adv_{ID}^{CUR}(C) + 2^{-\alpha+1}$$

Consider an UF-NMA adversary B perfectly simulating A's view in $Game_3$, while using RO' to simulate RO and a $2\kappa_m Q_H$-wise hash function to perfectly simulate RF. If (M^*, σ^*) is a valid forgery in $Game_3$, then it also so for our adversary B, and we get

$$\Pr[Game_3^A \Rightarrow 1] = Adv_{SIG}^{UF-NMA}(B)$$

Algorithm: *Game*$_3$

```
1  $(pk, sk) \leftarrow \text{IGen}(\text{par};$
2  $(\mu^*, \sigma^*) \leftarrow A^{|H\rangle, \text{Sign}_1}(pk);$
3  $\sigma^* = (W^*, Z^*);$
4  $c^* = \text{QRO}(W^*||\mu^*);$
5  if $c^* \neq RO(W^*||M^*)$ then
6      Return reject
7  else
8      if $\mu^* \notin \mathcal{M}$ and $V(pk, W^*, c^*, z^*)$ then
9          Return accept
10     else
11         Return reject
```

Algorithm: Sign_1

```
   Input: $\mu$
1  if $\mu \in \mathcal{M}$ then
2      Return $\perp$
3  $\mathcal{M} = \mathcal{M} \cup \mu;$
4  $(W_\mu, c_\mu, z_\mu) = \text{GetTrans}(\mu);$
5  Return $\sigma_\mu = (W\mu, Z_\mu)$
```

Algorithm: QRO

```
   Input: $W||\mu$
1  $(W_\mu, c_\mu, Z_\mu) = \text{GetTrans}(\mu);$
2  if $W = W_\mu$ then
3      Return $c = c_\mu$
4  else
5      Return $c = \text{RO}(w||\mu)$
```

Algorithm: GetTrans

```
   Input: $\mu$
1  $\kappa = 0;$
2  while $Z_\mu = \perp$ and $\kappa \leq \kappa_m$ do
3      $\kappa = \kappa + 1;$
4      $(W_\mu, c_\mu, Z_\mu) = \text{Sim}(pk; \text{RF}(\mu||\kappa));$
5  if $Z_\mu = \perp$ then
6      $(W_\mu, c_\mu, Z_\mu) = (\perp, \perp, \perp)$
7  Retunr $(W_\mu, c_\mu, Z_\mu)$
```

Figure 10: Game$_3$

Now we can put all of these equations together and we see that

$$Adv_{SIG}^{SUF-CMA_1}(A) \leq Adv_{SIG}^{UF-NMA}(B) + 2^{-\alpha+1} + adv_{ID}^{CUR}(C) + \kappa_m Q_S \cdot \epsilon_{zk}$$

□

Lemma 4.3 *If $(w_1, c, (\boldsymbol{z}, \boldsymbol{h}))$ and $(w_1, c, (\boldsymbol{z}', \boldsymbol{h}'))$ are such that $V(pk, w_1, c, (\boldsymbol{z}, \boldsymbol{h})) = V(pk, w_1, c, (\boldsymbol{z}', \boldsymbol{h}')) = 1$ and $(\boldsymbol{z}, \boldsymbol{h}) \neq (\boldsymbol{z}', \boldsymbol{h}')$, then there exist $\boldsymbol{v}, \boldsymbol{u}$ such that $||\boldsymbol{v}||_\infty < 2(\gamma_1 - \beta)$, $||\boldsymbol{u}||_\infty \leq 4\gamma_2 + 2$ such that $\boldsymbol{A}\boldsymbol{v} + \boldsymbol{u} = 0$.*

Proof Our conditions in the lemma implies

$$w_1 = \text{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$$
$$w_1 = \text{UseHint}_q(\mathbf{h}', \mathbf{Az}' - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$$

This again implies

$$||\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d - w_1 \cdot 2\gamma_2||_\infty \leq 2\gamma_2 + 1$$
$$||\mathbf{Az}' - c\mathbf{t}_1 \cdot 2^d - w_1 \cdot 2\gamma_2||_\infty \leq 2\gamma_2 + 1$$

Then the triangular inequality we get

$$\mathbf{A}(\mathbf{z} - \mathbf{z}') + \mathbf{u} = 0$$

for a $\mathbf{u}$ such that $||\mathbf{u}||_\infty \leq 4\gamma_2 + 2$ and $||\mathbf{z} - \mathbf{z}'||_\infty < 2(\gamma_1 - \beta)$.

□

Lemma 4.4 *For any quantum adversary A against UF-NMA security that issues at most Q_H queries to the quantum oracle QO, there exist quantum adversaries B and C such that*

$$Adv_{SIG}^{UF-NMA}(A) \leq Adv_{k,l,D}^{MLWE}(B) + Adv_{H,k,l+1,\zeta}^{SelfTargetMSIS}(C)$$

Proof When adversary C is given an $\mathbf{A}' = [\mathbf{I}|\mathbf{A}'']$ for $\mathbf{A}'' \in R_q^{k\times(l+1)}$, it can write $\mathbf{A}'' = [\mathbf{A}|\mathbf{t}]$ and then set the public key of the signature scheme to be $(\mathbf{A}, \mathbf{t})$. C then sends the public key to A. If the key is indistinguishable from the pk generated by our IGen, then A will return a signature of message M such that $||\mathbf{z}||_\infty < \gamma_1 - \beta$ and the verification equation

$$c = H(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d + \mathbf{u} \mid\mid M)$$

holds with probability $Adv_{H,k,l+1,\gamma}^{SelfTargetMSIS}(A)$. Here $||\mathbf{u}||_\infty \leq 2\gamma_2 + 1$. We know that $\mathbf{t} = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$ where $||\mathbf{t}_0||_\infty \leq 2^{d-1}$, hence we can rewrite the verification equation as

$$c = H(\mathbf{Az} - c\mathbf{t} + (c\mathbf{t}_0 + \mathbf{u}) \mid\mid M) = H(\mathbf{Az} - c\mathbf{t} + \mathbf{u}' \mid\mid M)$$

with $||\mathbf{u}'||_\infty \leq ||\mathbf{u}||_\infty + ||c\mathbf{t}_0||_\infty \leq 2\gamma_2 + 1 + 2^{d-1} \cdot \rho$. We now get a $\mathbf{y} \in R_q^{k\times(k+l+1)}$ such that $H(\mathbf{A}'\mathbf{y} \mid\mid M) = c$ where $y = \begin{bmatrix} \mathbf{r} \\ c \end{bmatrix}$ and $||\mathbf{y}||_\infty = ||\mathbf{r}||_\infty = \max\{\gamma_1 - \beta, 2\gamma_2 + 1 + 2^{d-1} \cdot \rho\}$.

□

Having proved these we are now ready to put them all together in order to prove the security of our signature scheme Dilithium.

Theorem 3 *If QRO is a quantum random oracle, the advantage of an adversary A for breaking SUF-CMA on Dilithium is*

$$Adv_{Dilithium}^{SUF-CMA}(A) \leq Adv_{k,l,D}^{MLWE}(B) + Adv_{H,k,l+1,\zeta}^{SelfTargetMSIS}(C) + Adv_{k,l,\zeta'}^{MSIS}(D) + 2^{-254}$$

for D a uniform distribution over S_η, and

$$\zeta = max\{\gamma_1 - \beta, 2\gamma_2 + 1 + 2^{d-1} \cdot 60\} \leq 4_{\gamma_2}$$
$$\zeta' = max\{2(\gamma_! - \beta), 4_{\beta_2} + 2\} \leq 4_{\gamma_2} + 2$$

Proof From Proposition 4.1 we have the following inequality

$$Adv_{SIG}^{SUF-CMA_1}(A) \leq Adv_{SIG}^{UF-NMA}(B) + 2^{-\alpha+1} + adv_{ID}^{CUR}(C) + \kappa_m Q_S \cdot \epsilon_{zk}$$

Lemma 4.3 implies that $Adv_{ID}^{CUR}(C) \leq Adv_{k,l,\zeta'}^{MSIS}(E)$ and it has been proven that the min-entropy α is greater than 255. Seen as this signature is only a simplification on Dilithium we can also conclude that $Adv_{Dilithium}^{SUF-CMA}(A) \leq Adv_{SIG}^{SUF-CMA}(A)$. Combining all of this gives us our final equation

$$Adv_{Dilithium}^{SUF-CMA}(A) \leq Adv_{k,l,D}^{MLWE}(B) + Adv_{H,k,l+1,\zeta}^{SelfTargetMSIS}(C) + Adv_{k,l,\zeta'}^{MSIS}(D) + 2^{-254}$$

□

5 Discussion

In this section we will discuss the differences between the schemes we presented in the previose section. We want to understand why we are not satisfied with Fiat-Shamir and Lyubashevsky and therefore have constructed Dilithium based on them. It is important to understand what improvements were added to Dilithium and how are they improving the scheme.

Fiat-Shamir, as stated before, is in itself not a quantum safe signature. However, the technique for converting an ID-scheme can be used on both post- and pre-quantum systems, and it is first and foremost this technique we want to illustrate. The method uses some random-looking function, often a hash function, to simulate the randomness of Victor's challenge. This method is, as we have seen, used in all three of our schemes. Lyubashevsky's scheme uses some random oracle to generate a random challenge. In Dilithium the function H is used.

H is undefined as a random oracle, but it concretely uses the SHAKE-256 hash function on the 384-bit string from m followed by the $k \times 128$-byte string from $\mathbf{w}_1$. The first 60 bits from the first 8 bytes of the output is put in $\mathbf{s} = (s_1, ..., s_{60})$, respectively, discarding the remaining 4 bits. Having $\mathbf{s}$ we then use rejection sampling to create c. We are then left with an element from R with 60 coefficients either 1 or -1 and the remaining being 0.

Dilithium uses two main hash functions, namely SHAKE-128 and SHAKE-256. H, as stated above, ExpandMask and CRH are all based on using SHAKE-256 to be randomn-looking. SHAKE-128 is only used in the construction of the matrix $\mathbf{A}$ in ExpandA. This function takes in the 256-bit element ρ, feeding it to SHAKE-128 to compute each individual element in the $k \times l$ matrix using rejection sampling in the order of our NTT domain representation. This matrix $\mathbf{A}$ has an equivalent role to h in Luybashevsky.

ExpandMask's purpose is to help create a random commitment. It computes each of the l coefficients in y independently by taking 32 bytes from K, 48 bytes from μ and a two byte representation of $l\kappa + i$ into SHAKE-256.

The collision resistante hash, CRH, is used two times in Dilithium. The first time it takes in the 236-bits from ρ followed by the $k \cdot 256 \cdot 9/8$ bits from the bit-string of t_1 and put them into SHAKE-256. The hash will then set the first 384-bit output from SHAKE-256 as its output tr. CRH is then used again with tr and μ as its input and feed it to SHAKE-256 and sets the first 384-bit output as its output m.

Scheme	weak	medium	recommended	very high
Lyubashevsky sk	16000	31000	49000	98000
Lyubashevsky vk	16000	31000	49000	98000
Lyubashevsky sig	49000	72000	119000	246000
Dilithium sk	17000	22000	28000	31000
Dilithium vk	7200	9500	12000	14000
Dilithium sig	11000	16000	22000	27000

Table 4: Estimate sizes in bits

A rather big difference between Luybashevsky and Dilithium is the construction of the keys and how this changes the signature and verification algorithms. The verification key is made much smaller, as can be seen in Table 4, by not including the matrix $\mathbf{A}$. Rather we let this matrix be computed individually in each of the algorithms using ExpandA. A consequence of this is also that we need more small variables for our keys, such as t_1, t_2. These are again made smaller by the function Power2round. We hide these further by using CRH on t_1. This is all done in such a way that we are able to provide a valid signature, and a consequence of this is that we get more computations in each algorithm. This is, however, not a problem, as the size of the matrix would have been greater than the size of all the variables combined, and the cost of computations are less than the cost of having the matrix in the keys.

All of the functions have specific purposes for improving Dilithium. ExpandA and CRH are both used to shorten down the keys. ExpandMask take something random looking and make it even more random. This might sound unnecessary, but it is hard to make something truly random, and having other parties find a pattern in our randomness can have grave consequences. Therefore we wish to hide our randomness in more randomness, as done with ExpandMask.

In order to reduce the size of the keys as wished with the above functions we need some helping algorithms. Namely the algorithms in Figure 6. We have stated before that these algorithms are used to produce hints so that we may recover high order bits later on. Power2round breaks up an element r into r_1, r_2 such that $r = r_2 \cdot 2^d + r_1$. We see that if we choose a representation of r_2 to be r_2', a non-negative integer between 0 and $q/2^d$, then the difference between $r_2 \cdot 2^d$ and $r_2' \cdot 2^d$ is most of the times less than 2^d. The instances where this is not the case create a problem for when we wish to produce a 1-bit hint, and this can lead to a high order bit change. This problem is avoided by choosing an α such that $\alpha \mid q-1$ and set $r = r_2 \cdot \alpha + r_1$. This procedure is handled by Decompose. Using these functions we can produce the functions MakeHint and UseHint to produce hints and recover the high order bits. HighBits and LowBits are mainly there to get r_1 and r_2 from Decompose as we wish.

We can examine the final size of our keys and signatures with different levels of security given and recommended for Luybashevsky and Dilithium [11, 25] in Table 4. The signature keys do not differ very much when using parameters with a weak security. In fact Dilithium actually has a bigger signature key in this case. However, when we increase the security of our schemes and hence adjust our parameters we see that the signature key of Dilithium becomes 1/3 the size of Luybashevsky. Likewise the verification key of Luybashevsky is seven times the size of the verification key of Dilithium and the signature over nine times bigger, with a very high security level. The increase in size makes Dilithium a lot more efficient as a signature. The improvements made to Dilithium gives it clear advantages.

Although the size reduction from Luybashevsky to Dilithium is rather big, it is worth noting that the keys to Dilithium are large compared to the keys of modern systems. For example, RSA has 2048-bit keys and ECC has 256-bit keys. While RSA and ECC are not quantum safe systems, they illustrate the key size of modern systems and thus how much more efficient they are. As long as the threat of quantum computers are no more then a theoretical threat, these key sizes illustrate the size and efficiency level post quantum cryptography need to reach in order to become competitive.

References

[1] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 10–19, 1998.

[2] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the second round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2020.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[4] Daniel J Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum rsa. In *International Workshop on Post-Quantum Cryptography*, pages 311–329. Springer, 2017.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions. `http://sponge.noekeon.org/CSF-0.1.pdf`, 2011.

[6] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak sha-3 submission. *Submission to NIST (Round 3)*, 6(7):16, 2011.

[7] Johannes Blömer and Jean-Pierre Seifert. On the complexity of computing short linearly independent vectors and short bases in a lattice. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 711–720, 1999.

[8] Shu-jen Chang, Ray Perlner, William E Burr, Meltem Sönmez Turan, John M Kelsey, Souradyuti Paul, and Lawrence E Bassham. Third-round report of the sha-3 cryptographic hash algorithm competition. *NIST Interagency Report*, 7896:121, 2012.

[9] Avinash Dash, Deepankar Sarmah, Bikash K Behera, and Prasanta K Panigrahi. Exact search algorithm to factorize large biprimes and a triprime on ibm quantum computer. *arXiv preprint arXiv:1805.10478*, 2018.

[10] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[11] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[12] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426, 1990.

[13] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. *Advances in Cryptology — CRYPTO' 86*, pages 186–194, 1987.

[14] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36, 2018.

[15] Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[16] Kristian Gjøsteen. Public key encryption, 2019 (accessed November 26, 2020). `https://wiki.math.ntnu.no/_media/tma4160/pke.pdf`.

[17] Kristian Gjøsteen. Zero knowledge, 2020. not published.

[18] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and J-P Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[19] Shafi Goldwasser and Daniele Micciancio. Complexity of lattice problems: a cryptographic perspective, 2002.

[20] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM (JACM)*, 52(5):789–808, 2005.

[21] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 552–586. Springer, 2018.

[22] Thijs Laarhoven, Joop van de Pol, and Benne de Weger. Solving hard lattice problems and the security of lattice-based cryptosystems. *IACR Cryptol. ePrint Arch.*, 2012:533, 2012.

[23] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[24] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. *ICALP (2)*, pages 144–155, 2006.

[25] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. *ASIACRYPT*, 2009.

[26] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[27] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

[28] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[29] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 387–398. Springer, 1996.

[30] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[31] NIST Sha. standard: Permutation-based hash and extendable-output functions. federal information processing standards publication 202, 2015.

[32] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[33] Daniel Smith-Tone and Ray Perlner. Rainbow band separation is better than we thought. *Cryptology ePrint Archive preprint*, 2020.

[34] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 617–635. Springer, 2009.

NTNU

Kunnskap for en bedre verden