Chapter 9

Post-Quantum Cryptography

In this chapter, we discuss several techniques for creating public-key cryptosystems and signature schemes in the setting of post-quantum cryptography. We include lattice-based cryptography (specifically *NTRU* and cryptography based on the **Learning With Errors** problem), code-based cryptography, multivariate cryptography, and hash-based signature schemes.

9.1 Introduction

The two previous chapters have dealt with public-key cryptography based on the presumed difficulty of the **Factoring** and **Discrete Logarithm** problems, respectively. However, there has been increased interest, especially in recent years, in developing public-key cryptosystems based on other underlying computational problems. One specific motivation for this interest is the ongoing research in ***quantum computing*** and the possible impact it might have on existing cryptographic schemes, in particular, public-key cryptography based on the **Factoring** and **Discrete Logarithm** problems.

Here is a useful high-level explanation of the basics of quantum computing:

> A traditional computer uses long strings of "bits," which encode either a zero or a one. A quantum computer, on the other hand, uses ***quantum bits***, or ***qubits***. What's the difference? Well a qubit is a quantum system that encodes the zero and the one into two distinguishable quantum states. But, because qubits behave quantumly, we can capitalize on the phenomena of ***superposition*** and ***entanglement***. Superposition is essentially the ability of a quantum system to be in multiple states at the same time—that is, something can be "here" and "there," or "up" and "down" at the same time. Entanglement is an extremely strong correlation that exists between quantum particles—so strong, in fact, that two or more quantum particles can be inextricably linked in perfect unison, even if separated by great distances. Thanks to superposition and entanglement, a quantum computer can process a vast number of calculations simultaneously. Think of it this way: whereas a classical computer works with ones and zeros, a quantum computer will have

1970s. The advantage of quantum cryptography is that it allows the construction of unconditionally secure schemes that cannot exist in the setting of ***classical cryptography***. However, quantum cryptography is a topic that we do not cover in this book.

The potential impact of quantum computers on secret-key cryptography appears to be much less drastic than on public-key cryptography. The main attack method that could be carried out by a quantum computer is based on ***Grover's Algorithm***. Roughly speaking, this permits certain types of exhaustive searches that would require time $O(m)$ on a "classical" (i.e., nonquantum) computer to be carried out in time $O(\sqrt{m})$ on a quantum computer. What this means is that a secure secret-key cryptosystem having key length ℓ should be replaced by one having key length 2ℓ in order to remain secure against a quantum computer. This is because an exhaustive search of an ℓ-bit key on a classical computer takes time $O(2^\ell)$, and an exhaustive search of a 2ℓ-bit key on a quantum computer takes time $O(\sqrt{2^{2\ell}})$, which is the same as $O(2^\ell)$ because $\sqrt{2^{2\ell}} = (2^{2\ell})^{1/2} = 2^\ell$.

There have been several interesting approaches to post-quantum cryptography that have been investigated in recent years. These include the following:

lattice-based cryptography
: We discuss *NTRUEncrypt* in Section 9.2.1; this system is defined using arithmetic in certain polynomial rings. Other examples of lattice-based cryptography are based on the **Learning With Errors** problem, which originated in the field of machine learning. We present a simple cryptosystem based on this problem, the *Regev Cryptosystem*, in Section 9.2.3.

code-based cryptography
: Section 9.3 gives a short description of the *McEliece Cryptosystem*, which involves error-correcting codes.

multivariate cryptography
: Techniques of multivariate cryptography have been considered in the context of cryptosystems (*Hidden Field Equations*; see Section 9.4.1) as well as signature schemes (*Oil and Vinegar*; see Section 9.4.2) .

hash-based cryptography
: Hash-based cryptography is used primarily for signature schemes; see Section 9.5.

isogeny-based cryptography
: The idea of isogeny-based cryptography is based on certain morphisms between different elliptic curves. However, we do not discuss these techniques in this book.

In any discussion of post-quantum cryptography, it should be pointed out that the proposed techniques are not proven to be immune to attacks by quantum computers. Rather, the approach is to utilize problems that, at present, are not susceptible to quantum attacks based on currently known algorithms.

the advantage of using ones, zeros and "superpositions" of ones and zeros."[1]

Explaining these ideas in detail would require considerable background, so we are not going to attempt to discuss quantum computing except in the most broad terms. Historically, the basic idea of quantum computing dates back to at least 1980, and the relevance of quantum computing became evident with the publication of SHOR'S ALGORITHM in 1994.

Before we delve into the implications of SHOR'S ALGORITHM, we should mention that the development of a practical quantum computer appears to be some years in the future. Despite intense research during the last twenty years, construction of a scalable, fault-tolerant quantum computer has not been achieved yet. However, some experts have expressed the opinion that such a computer has a reasonable chance of being constructed by 2030 or thereabouts. More precisely, Mike Mosca, a leading researcher in quantum computing, predicted in 2016 that there is a one-in-seven chance that a quantum computer would be able to factor a 2048-bit RSA modulus by 2026, and a 50% probability that this would be achieved by 2031.

SHOR'S ALGORITHM shows that integers could be factored quickly using a quantum computer.[2] More precisely, SHOR'S ALGORITHM has complexity $O((\log n)^2(\log\log n)(\log\log\log n))$ to factor a positive integer n. This is a polynomial-time algorithm as a function of the "size" of n, which is $\log n$. SHOR'S ALGORITHM can also be used to solve the **Discrete Logarithm** problem efficiently (i.e., in polynomial time).

So, the consequence of SHOR'S ALGORITHM is the following: if a scalable, fault-tolerant quantum computer can be built, then public-key cryptography based on **Factoring** and **Discrete Logarithm** problems is irretrievably broken. Based on this possible scenario, researchers have been studying potential ways of constructing public-key cryptosystems based on different computational problems, which hopefully would not be susceptible to attacks carried out by quantum computers. The term ***post-quantum cryptography*** is used to describe such cryptographic schemes. More generally, the phrase "post-quantum cryptography" can also apply to other types of public-key primitives (such as signature schemes, for example, which we introduced in Chapter 8).

We should also take a moment to clarify the distinction between quantum cryptography and post-quantum cryptography. ***Quantum cryptography*** refers to cryptographic algorithms or primitives that rely on quantum mechanical techniques for their implementation. Quantum cryptography includes algorithms for ***quantum key distribution*** and ***quantum bit commitment***, among other things. The basic idea of quantum cryptography dates back to Stephen Wiesner's work in the early

[1] `https://uwaterloo.ca/institute-for-quantum-computing/quantum-computing-101#What-is-quantum-computing`

[2] The idea of this algorithm is to compute the order of the element 3 (or some other small integer) in $\mathbb{Z}_n^*$. This order will be a divisor of $\phi(n)$ and "usually" it will lead to the determination of a nontrivial factor of n.

to be

$$\begin{aligned}
\mathbf{y} &= \mathbf{x}G' + \mathbf{e} \\
&= (1,1,0,1)\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} + (0,0,0,0,1,0,0) \\
&= (0,1,1,0,0,1,0) + (0,0,0,0,1,0,0) \\
&= (0,1,1,0,1,1,0).
\end{aligned}$$

When Bob receives the ciphertext $\mathbf{y}$, he first computes

$$\begin{aligned}
\mathbf{y}_1 &= \mathbf{y}P^{-1} \\
&= (0,1,1,0,1,1,0)\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \\
&= (1,0,0,0,1,1,1).
\end{aligned}$$

Next, he decodes $\mathbf{y}_1$ to get $\mathbf{x}_1 = (1,0,0,0,1,1,0)$. It is worth noting that $\mathbf{e}_1 \neq \mathbf{e}$ due to the multiplication by P^{-1}. However, since P is a permutation matrix, the multiplication only changes the position of the error(s).

Next, Bob forms $\mathbf{x}_0 = (1,0,0,0)$ (the first four components of $\mathbf{x}_1$).

Finally, Bob calculates

$$\begin{aligned}
\mathbf{x} &= \mathbf{x}_0 S^{-1} \\
&= (1,0,0,0)\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \\
&= (1,1,0,1).
\end{aligned}$$

This is indeed the plaintext that Alice encrypted. □

9.4 Multivariate Cryptography

Another example of a problem suggested for use in the design of post-quantum cryptosystems is that of finding solutions to large systems of quadratic equations in many variables over a finite field. This is known as the **Multivariate Quadratic**

$2, \ldots, \lfloor (d-1)/2 \rfloor$. If, at any time, we have $H\mathbf{e}^T = \mathbf{s}$ for a candidate error vector $\mathbf{e}$, then we decode $\mathbf{r}$ to $\mathbf{r} - \mathbf{e}$ and quit. If this equation is never satisfied, then we conclude that more than $\lfloor (d-1)/2 \rfloor$ errors have occurred during transmission.

Using this approach, we can decode a received vector in at most

$$1 + \binom{n}{1} + \cdots + \binom{n}{\lfloor (d-1)/2 \rfloor}$$

steps.

This method works on any linear code. Further, for certain specific types of codes, the decoding procedure can be speeded up. However, nearest neighbor decoding is in fact an NP-hard problem. Thus, no polynomial-time algorithm is known for the general problem of nearest neighbor decoding.

It turns out that it is possible to identify an "easy" special case of the decoding problem and then disguise it so that it looks like a "difficult" general case of the problem. It would take us too long to go into the theory here, so we will just summarize the results. The "easy" special case that was suggested by McEliece is to use a code from a class of codes known as the ***Goppa codes***. These codes do in fact have efficient decoding algorithms. Also, they are easy to generate and there are a large number of inequivalent Goppa codes with the same parameters.

The parameters of the Goppa codes have the form $n = 2^m$, $d = 2t + 1$, and $k = n - mt$ for an integer t. For a practical implementation of the public-key cryptosystem, McEliece originally suggested taking $m = 10$ and $t = 50$. This gives rise to a Goppa code that is a linear $[1024, 524, 101]$ code. Each plaintext is a binary 524-tuple, and each ciphertext is a binary 1024-tuple. The public key is a 524×1024 binary matrix. However, current recommended parameter sizes are considerably larger than these. For example, a 2008 study by Bernstein, Lange, and Peters recommended taking $m = 11$ and $t = 27$, which utilizes a linear $[2048, 1751, 55]$ Goppa code, for a minimum acceptable level of security.

A description of the *McEliece Cryptosystem* is given in Cryptosystem 9.3. We present a toy example to illustrate the encoding and decoding procedures.

Example 9.3 The matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

is a generating matrix for a linear $[7, 4, 3]$ code, known as a ***Hamming code***. Suppose Bob chooses the matrices

$$S = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Shubha · Puttaraju

Cryptosystem 9.3: *McEliece Cryptosystem*

Let G be a generating matrix for a linear $[n,k,d]$ Goppa code $\mathbf{C}$, where $n = 2^m$, $d = 2t+1$, and $k = n - mt$. Let S be a $k \times k$ matrix that is invertible over $\mathbb{Z}_2$, let P be an $n \times n$ permutation matrix, and let $G' = SGP$. Let $\mathcal{P} = (\mathbb{Z}_2)^k$, $\mathcal{C} = (\mathbb{Z}_2)^n$, and let

$$\mathcal{K} = \{(G, S, P, G')\},$$

where G, S, P, and G' are constructed as described above. The matrix G' is the public key and G, S, and P comprise the private key.

For a public key G', a plaintext $\mathbf{x} \in (\mathbb{Z}_2)^k$ is encrypted by computing

$$\mathbf{y} = \mathbf{x}G' + \mathbf{e},$$

where $\mathbf{e} \in (\mathbb{Z}_2)^n$ is a random error vector of weight t.

A ciphertext $\mathbf{y} \in (\mathbb{Z}_2)^n$ is decrypted by means of the following operations:

1. Compute $\mathbf{y}_1 = \mathbf{y}P^{-1}$.
2. Decode $\mathbf{y}_1$, obtaining $\mathbf{y}_1 = \mathbf{x}_1 + \mathbf{e}_1$, where $\mathbf{x}_1 \in \mathbf{C}$.
3. Compute $\mathbf{x}_0 \in (\mathbb{Z}_2)^k$ such that $\mathbf{x}_0 G = \mathbf{x}_1$.
4. Compute $\mathbf{x} = \mathbf{x}_0 S^{-1}$.

and

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Then, the public generating matrix is

$$G' = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Now, suppose Alice encrypts the plaintext $\mathbf{x} = (1,1,0,1)$ using the vector $\mathbf{e} = (0,0,0,0,1,0,0)$ as the random error vector of weight 1. The ciphertext is computed

Cryptosystem 9.2: *Regev Cryptosystem*

Let n and m be integers and let q be a prime. Let $\mathbf{E}$ be a discrete random variable defined on $\mathbb{Z}_q$.

The private key is an element $\mathbf{s} \in (\mathbb{Z}_q)^n$.

The public key consists of m samples $(\mathbf{a^i}, b^i)$ where $\mathbf{a^i}$ is drawn uniformly from $\mathbb{Z}_q$ and b^i is taken to be $b^i = \mathbf{E} + \sum_{j=1}^{n} a_j^i s_j$.

To encrypt a one-bit message x, choose a random subset $S \subseteq \{1, 2, \ldots, m\}$. The ciphertext y is given by

$$y = \begin{cases} (\sum_{i \in S} \mathbf{a^i}, \sum_{i \in S} b^i) & \text{if } x = 0, \\ (\sum_{i \in S} \mathbf{a^i}, \lfloor \frac{q}{2} \rfloor + \sum_{i \in S} b^i) & \text{if } x = 1. \end{cases}$$

To decrypt a ciphertext $(\mathbf{a}, b)$, compute the quantity $b - \sum_{j=1}^{m} a_j s_j$. The decrypted message is 0 if the result is closer to 0 than $\lfloor q/2 \rfloor$ and 1 otherwise.

Cryptosystem 9.2 is not practical due to the large overhead required to encrypt a single bit. There exist more efficient cryptosystems based on **LWE** (including ones that are secure against chosen ciphertext attacks) as well as many other cryptographic primitives. However, these schemes tend to require large keys. One way to reduce the size of the public key in Cryptosystem 9.2 is to replace the uniformly generated $\mathbf{a^i}$ with a more structured set of elements of $(\mathbb{Z}_q)^n$. This idea has led to the development of cryptosystems based on a variant of **LWE** known as **ring-LWE**. This approach permits the construction of more efficient schemes, but addressing the question of how to determine suitable error distributions is even more subtle than in the case of **LWE**.

9.3 Code-based Cryptography and the McEliece Cryptosystem

The ***NP-complete problems*** comprise a large class of decision problems (i.e., problems that have a yes/no answer) that are believed to be impossible to solve in polynomial time. The ***NP-hard problems*** are a class of problems, which may or may not be decision problems, that are at least as difficult to solve as the NP-complete problems.

In the *McEliece Cryptosystem*, decryption is an easy special case of an NP-hard problem, disguised so that it looks like a (presumably difficult) general instance of the problem. In this system, the NP-hard problem that is employed is related to decoding a general linear (binary) error-correcting code. However, for many

special classes of codes, polynomial-time algorithms are known to exist. One such class of codes is used as the basis of the *McEliece Cryptosystem*.

We begin with some essential definitions. First we define the notion of a linear code and a generating matrix.

Definition 9.2: Let k, n be positive integers, $k \leq n$. A ***linear code*** is a k-dimensional subspace of $(\mathbb{Z}_2)^n$, the vector space of all binary n-tuples. A ***linear*** $[n,k]$ ***code***, $\mathbf{C}$, is a k-dimensional subspace of $(\mathbb{Z}_2)^n$.

A ***generating matrix*** for a linear $[n,k]$ code, $\mathbf{C}$, is a $k \times n$ binary matrix whose rows form a basis for $\mathbf{C}$.

Next, we define the distance of a (linear) code.

Definition 9.3: Let $\mathbf{x}, \mathbf{y} \in (\mathbb{Z}_2)^n$, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$. Define the ***Hamming distance***

$$\mathbf{dist}(\mathbf{x}, \mathbf{y}) = |\{i : 1 \leq i \leq n, x_i \neq y_i\}|,$$

i.e., the number of co-ordinates in which $\mathbf{x}$ and $\mathbf{y}$ differ.

Let $\mathbf{C}$ be a linear $[n,k]$ code. Define the ***distance*** of $\mathbf{C}$ to be the quantity

$$\mathbf{dist}(\mathbf{C}) = \min\{\mathbf{dist}(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathbf{C}, \mathbf{x} \neq \mathbf{y}\}.$$

A ***linear*** $[n,k,d]$ ***code*** is a linear $[n,k]$ code, say $\mathbf{C}$, in which $\mathbf{dist}(\mathbf{C}) \geq d$.

Finally, we define the dual code (of a linear code) and the parity-check matrix.

Definition 9.4: Two vectors $\mathbf{x}, \mathbf{y} \in (\mathbb{Z}_2)^n$, say $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, are ***orthogonal*** if

$$\sum_{i=1}^{n} x_i y_i \equiv 0 \pmod 2.$$

The ***orthogonal complement*** of a linear $[n,k,d]$ code, $\mathbf{C}$, consists of all the vectors that are orthogonal to all the vectors in $\mathbf{C}$. This set of vectors is denoted by $\mathbf{C}^{\perp}$ and it is called the ***dual code*** to $\mathbf{C}$.

A ***parity-check matrix*** for a linear $[n,k,d]$ code $\mathbf{C}$ having generating matrix G is a generating matrix H for $\mathbf{C}^{\perp}$. This matrix H is an $(n-k)$ by n matrix. (Stated another way, the rows of H are linearly independent vectors, and GH^T is a k by $n-k$ matrix of zeroes.)

The purpose of an error-correcting code is to correct random errors that occur in the transmission of (binary) data through a noisy channel. Briefly, this is done as follows. Let G be a generating matrix for a linear $[n,k,d]$ code. Suppose $\mathbf{x}$ is the

binary k-tuple we wish to transmit. Then we encode $\mathbf{x}$ as the n-tuple $\mathbf{y} = \mathbf{x}G$ and we transmit $\mathbf{y}$ through the channel.

Now, suppose Bob receives the n-tuple $\mathbf{r}$, which may not be the same as $\mathbf{y}$. He will decode $\mathbf{r}$ using the strategy of "nearest neighbor decoding." The idea is that Bob finds a codeword $\mathbf{y}' \neq \mathbf{r}$ that has minimum distance to $\mathbf{r}$. Such a codeword will be called a ***nearest neighbor*** to $\mathbf{r}$ and it will be denoted as by $\mathbf{nn}(\mathbf{r})$ (note that it is possible that there might be more than one nearest neighbor). The process of computing $\mathbf{nn}(\mathbf{r})$ is called ***nearest neighbor decoding***.

After decoding $\mathbf{r}$ to $\mathbf{y}' = \mathbf{nn}(\mathbf{r})$, Bob would determine the k-tuple $\mathbf{x}'$ such that $\mathbf{y}' = \mathbf{x}'G$. Bob is hoping that $\mathbf{y}' = \mathbf{y}$, so $\mathbf{x}' = \mathbf{x}$ (i.e., he is hoping that any transmission errors have been corrected).

It is fairly easy to show that if at most $(d-1)/2$ errors occurred during transmission, then nearest neighbor decoding does in fact correct all the errors. In this case, any received vector $\mathbf{r}$ will have a unique nearest neighbor, and $\mathbf{nn}(\mathbf{r}) = \mathbf{y}$.

Let us think about how nearest neighbor decoding would be done in practice. The number of possible codewords is equal to 2^k. If Bob compares $\mathbf{r}$ to every codeword, then he will have to examine 2^k vectors, which is an exponentially large number compared to k. In other words, this obvious decoding algorithm is not a polynomial-time algorithm.

Another approach, which forms the basis for many practical decoding algorithms, is based on the idea of a syndrome. Suppose $\mathbf{C}$ is a linear $[n,k]$ code having parity-check matrix H. Given a vector $\mathbf{r} \in (\mathbb{Z}_2)^n$, we define the ***syndrome*** of $\mathbf{r}$ to be Hr^T. A syndrome is a column vector with $n-k$ components.

The following basic result can be proven using straightforward techniques from linear algebra.

THEOREM 9.1 *Suppose* $\mathbf{C}$ *is a linear* $[n,k]$ *code with parity-check matrix* H. *Then* $\mathbf{x} \in (\mathbb{Z}_2)^n$ *is a codeword if and only if*

$$H\mathbf{x}^T = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Further, if $\mathbf{x} \in \mathbf{C}$, $\mathbf{e} \in (\mathbb{Z}_2)^n$ *and we define* $\mathbf{r} = \mathbf{x} + \mathbf{e}$, *then* $H\mathbf{r}^T = H\mathbf{e}^T$.

Think of $\mathbf{e}$ as being the vector of errors that occur during transmission of a codeword $\mathbf{x}$. Then $\mathbf{r}$ represents the vector that is received. The above theorem is saying that the syndrome depends only on the errors, and not on the particular codeword that was transmitted.

This suggests the following approach to decoding, known as ***syndrome decoding***: First, compute $\mathbf{s} = H\mathbf{r}^T$. If $\mathbf{s}$ is a vector of zeroes, then decode $\mathbf{r}$ as $\mathbf{r}$. If not, then generate all possible error vectors of weight 1 in turn, where the ***weight*** of a vector is the number of nonzero components it contains. For each such error vector $\mathbf{e}$, compute $H\mathbf{e}^T$. If, for any of these vectors $\mathbf{e}$, it holds that $H\mathbf{e}^T = \mathbf{s}$, then decode $\mathbf{r}$ to $\mathbf{r} - \mathbf{e}$. Otherwise, continue on to generate all error vectors of weight

Reducing the coefficients of $a(x)$ modulo 3 yields

$$x^{15} - x^{12} + x^7 - 1,$$

which is the plaintext.

In this example, decryption yielded the correct plaintext because (9.2) is satisfied, as can easily be verified. □

There are a couple of additional conditions on the parameters that we should mention. First, it is usually recommended that each of $\mathbf{F}$, $\mathbf{G}$, $\mathbf{r}$, and $\mathbf{m}$ have (roughly) one third of their coefficients equal to each of 0, -1, and 1. These requirements are related to the security of the scheme. The second condition is that q should be large compared to N, so the decryption condition (9.2) holds with certainty, or at least with very high probability. The parameter choices mentioned above ensure that this will be the case.

We briefly discuss the decryption operation in a bit more detail now. Suppose we focus on a specific co-ordinate of $\mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m}$, say the ith co-ordinate. This would be computed as the sum of the ith co-ordinates of $\mathbf{r} \star \mathbf{g}$ and $\mathbf{f} \star \mathbf{m}$, each of which are obtained from the convolution formula (9.1). First, let's focus on a co-ordinate of $\mathbf{r} \star \mathbf{g}$. The formula (9.1) is the sum of N terms. The N-tuple $\mathbf{r}$ has (approximately) $N/3$ co-ordinates equal to each of 1, 0, and -1, and $\mathbf{g}$ has (approximately) $N/3$ co-ordinates equal to each of p, 0, and $-p$. The maximum value taken on by a particular co-ordinate of $\mathbf{r} \star \mathbf{g}$ would therefore be

$$\frac{N}{3}(p \times 1 + (-p) \times (-1)) = \frac{2Np}{3}.$$

The maximum value of a co-ordinate of $\mathbf{f} \star \mathbf{m}$ is also $2Np/3$. So the maximum value of a co-ordinate of $\mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m}$ is $4Np/3 = 4N = 1604$, using the values $p = 3$ and $N = 401$. Similarly, the minimum value is -1604. So the maximum and minimum values are outside the interval

$$\left[-\frac{q-1}{2}, \frac{q}{2}\right] = [-1023, 1024],$$

which means that a decryption error is possible. However, it is very unlikely that all the co-ordinates "line up" so the maximum or minimum is actually achieved. A more detailed analysis shows that the probability of a decryption error is very small.

9.2.2 Lattices and the Security of NTRU

We mentioned that the security of *NTRUEncrypt* is related to certain lattice problems. However, before discussing security, we need to develop some of the basic theory of lattices.

A lattice is very similar to a vector space. A ***real vector space*** can be defined by starting with a ***basis***, which is a set of linearly independent vectors in $\mathbb{R}^n$ for some integer n. The vector space generated by the given basis consists of all linear

Let S be the affine transformation that maps

$$(x_1, x_2, x_3, x_4) \mapsto (x_2 + x_4 + 1, x_1 + x_4 + 1, x_2 + x_3 + x_4, x_1 + x_2 + x_3 + x_4).$$

Using (9.9). it can be shown that S^{-1} is the transformation that maps

$$(y_1, y_2, y_3, y_4) \mapsto (y_3 + y_4, y_1 + y_2 + y_3 + y_4, y_1 + y_3 + 1, y_2 + y_3 + y_4 + 1).$$

Applying (9.11), the polynomials f_1^{pub} and f_2^{pub} are given by

$$\begin{aligned} f_1^{\text{pub}}(x_1, x_2, x_3, x_4) &= x_1x_3 + x_3x_4 + x_2 + 1 \quad \text{and} \\ f_2^{\text{pub}}(x_1, x_2, x_3, x_4) &= x_1x_2 + x_1x_3 + x_2x_3 + x_2x_4 + x_1 + x_2 + x_4 + 1. \end{aligned}$$

Suppose we wish to sign the message $(0, 1)$. This requires solving the system of equations $f_1(x_1, x_2, x_3, x_4) = 0$ and $f_2(x_1, x_2, x_3, x_4) = 1$. To do this, we guess values for the vinegar variables, say $x_1 = 0$ and $x_2 = 1$. Then the equations we need to solve become

$$\begin{aligned} f_1(0, 1, x_3, x_4) &= x_3 + x_4 = 0, \\ f_2(0, 1, x_3, x_4) &= x_4 + 1 = 1. \end{aligned}$$

This system has the unique solution $x_3 = x_4 = 0$, and hence we conclude that $(0, 1, 0, 0)$ is a solution to our original system of equations. The required signature is then given by $S^{-1}(0, 1, 0, 0) = (0, 1, 1, 0)$.

To verify that $(0, 1, 1, 0)$ is a valid signature for the message $(0, 1)$, we simply evaluate $f_1^{\text{pub}}(0, 1, 1, 0)$ and $f_2^{\text{pub}}(0, 1, 1, 0)$, which gives the results 0 and 1 respectively. □

As in the case of *HFE*, the *Oil and Vinegar Signature Scheme* has been broken, as the affine transformation does not adequately hide the very structured nature of the original system of equations. Suggested approaches to improving the security of this scheme have included increasing the number of vinegar variables (the so-called *Unbalanced Oil and Vinegar Signature Scheme*). One set of parameters, proposed by Kipnis, Patarin, and Goubin in 2009, uses the field $\mathbb{F}_2$ with 64 oil variables and 128 vinegar variables.

9.5 Hash-based Signature Schemes

In this section, we describe some nice techniques to construct signature schemes based only on hash functions (or possibly even one-way functions). Thus these signature schemes are of considerable interest in the setting of post-quantum cryptography.

Cryptosystem 9.6: *Lamport Signature Scheme*

Let k be a positive integer and let $\mathcal{P} = \{0,1\}^k$. Suppose $f : Y \to Z$ is a one-way function (in practice, the function f would probably be a secure hash function). Let $\mathcal{A} = Y^k$. Let $y_{i,j} \in Y$ be chosen at random, $1 \leq i \leq k$, $j = 0, 1$, and let $z_{i,j} = f(y_{i,j})$, $1 \leq i \leq k$, $j = 0, 1$. The key K consists of the $2k$ y's and the $2k$ z's. The y's are the private key while the z's are the public key.

For $K = (y_{i,j}, z_{i,j} : 1 \leq i \leq k, j = 0, 1)$, define

$$\mathbf{sig}_K(x_1, \ldots, x_k) = (y_{1,x_1}, \ldots, y_{k,x_k}).$$

A signature $(a_1, \ldots, a_k)$ on the message $(x_1, \ldots, x_k)$ is verified as follows:

$$\mathbf{ver}_K((x_1, \ldots, x_k), (a_1, \ldots, a_k)) = \text{true} \Leftrightarrow f(a_i) = z_{i,x_i}, 1 \leq i \leq k.$$

9.5.1 Lamport Signature Scheme

First, we discuss a conceptually simple way to construct a provably secure one-time signature scheme from a one-way function. (A signature scheme is a ***one-time signature scheme*** if it is secure when only one message is signed. The signature can be verified an arbitrary number of times, of course.) The description of the scheme, which is known as the *Lamport Signature Scheme*, is given in Cryptosystem 9.6. This scheme was published in 1979, so it is one of the earliest examples of a signature scheme.

Informally, this is how the system works. A message to be signed is a binary k-tuple. In order to not have to worry about the length of the message, we assume that the value of k is fixed ahead of time.

Each bit of the message is signed individually. If the ith bit of the message equals j (where $j \in \{0,1\}$), then the ith element of the signature is the value $y_{i,j}$, which is a preimage of the public key value $z_{i,j}$. The verification consists simply of checking that each element in the signature is a preimage of the public key element $z_{i,j}$ that corresponds to the ith bit of the message. This can be done using the public function f.

We illustrate the scheme by considering one possible implementation using the exponentiation function $f(x) = \alpha^x \bmod p$, where α is a primitive element modulo p. Here $f : \{0, \ldots, p-2\} \to \mathbb{Z}_p{}^*$. We present a toy example to demonstrate the computations that take place in the scheme.

Example 9.9 7879 is prime and 3 is a primitive element in $\mathbb{Z}_{7879}{}^*$. Define

$$f(x) = 3^x \bmod 7879.$$

Suppose $k = 3$, and Alice chooses the six (secret) random numbers

$$\begin{aligned} y_{1,0} &= 5831 \\ y_{1,1} &= 735 \\ y_{2,0} &= 803 \\ y_{2,1} &= 2467 \\ y_{3,0} &= 4285 \\ y_{3,1} &= 6449. \end{aligned}$$

Then Alice computes the images of these six y's under the function f:

$$\begin{aligned} z_{1,0} &= 2009 \\ z_{1,1} &= 3810 \\ z_{2,0} &= 4672 \\ z_{2,1} &= 4721 \\ z_{3,0} &= 268 \\ z_{3,1} &= 5731. \end{aligned}$$

These z's are published. Now, suppose Alice wants to sign the message

$$x = (1, 1, 0).$$

The signature for x is

$$(y_{1,1}, y_{2,1}, y_{3,0}) = (735, 2467, 4285).$$

To verify this signature, it suffices to compute the following:

$$\begin{aligned} 3^{735} \bmod 7879 &= 3810 \\ 3^{2467} \bmod 7879 &= 4721 \\ 3^{4285} \bmod 7879 &= 268. \end{aligned}$$

Hence, the signature is verified. □

We argue that, if Oscar sees one message and its signature, then he will be unable to forge a signature on a second message. Suppose that $(x_1, \ldots, x_k)$ is a message and $(y_{1,x_1}, \ldots, y_{k,x_k})$ is its signature. Now suppose Oscar tries to sign the new message $(x'_1, \ldots, x'_k)$. Since this message is different from the first message, there is at least one co-ordinate i such that $x'_i \neq x_i$. Signing the new message requires computing a value a such that $f(a) = z_{i,x'_i}$. Since Oscar has not seen a preimage of z_{i,x'_i}, and f is a one-way function, he is unable to find a value a which would could be used in a valid signature for $(x'_1, \ldots, x'_k)$.

However, this signature scheme can be used to sign only one message securely. Given signatures on two different messages, it is an easy matter for Oscar to construct signatures for another message different from the first two (unless the first two messages differ in exactly one bit).

For example, suppose the messages $(0,1,1)$ and $(1,0,1)$ are both signed using the same key. The message $(0,1,1)$ has as its signature the triple $(y_{1,0}, y_{2,1}, y_{3,1})$, and the message $(1,0,1)$ is signed with $(y_{1,1}, y_{2,0}, y_{3,1})$. Given these two signatures, Oscar can manufacture signatures for the messages $(1,1,1)$ (namely, $(y_{1,1}, y_{2,1}, y_{3,1})$) and $(0,0,1)$ (namely, $(y_{1,0}, y_{2,0}, y_{3,1})$).

9.5.2 Winternitz Signature Scheme

The *Lamport Signature Scheme*, as described in Section 9.5.1, has a very large key size. To sign a k-bit message, we require a public key consisting of $2k$ values $z_{i,j}$ from the set Z. Since these z-values are probably outputs of a secure hash function, they would each be least 224 bits in length (for example, if we used the hash function *SHA3-224*). The *Winternitz Signature Scheme* provides a significant reduction in key size, by allowing multiple bits to be signed by each application of the one-way function f.

We first present the basic idea, which, however, is not secure. Then we describe how to fix the security problem.

We will sign w bits at a time, where w is a pre-specified parameter. Suppose f is a secure hash function. To illustrate the basic idea, let's fix $w = 3$ for the time being. Suppose a random value y_0 is chosen and we compute the ***hash chain***

$$y^0 \to y^1 \to y^2 \to y^3 \to y^4 \to y^5 \to y^6 \to y^7 \to z$$

according to the rules $y^j = f(y^{j-1})$ for $1 \le j \le 7$, and $z = f(y_7)$. We can equivalently define $y^j = f^j(y_0)$ for $1 \le j \le 7$, and $z = f^8(y_0)$, where f^j denotes j applications of the function f. The value z would be the public key for this hash chain. In general, the hash chain would consist of $2^w + 1$ values, namely, $y^0, \ldots, y^{2^w-1}, z$.

For a k-bit message, we would construct $\ell = k/w$ hash chains (let's assume that k is a multiple of w, for convenience). Denote the initial values in these hash chains by $y_1^0, y_2^0, \ldots, y_\ell^0$. These initial values comprise the private key.

Now consider a message $(x_1, \ldots, x_\ell)$, where each x_i is a binary w-tuple. Thus we can view each x_i as an integer between 0 and $2^w - 1$ (inclusive). As a first attempt at a signature, consider releasing the values $a_i = y_i^{x_i} = f^{x_i}(y_i)$ for $i = 1, \ldots, \ell$ as a signature. (Note that we do not need to store the entire hash chains; we can compute the a_i's, as needed, from the initial values.) Then, to verify a given a_i, it suffices to check that $f^{2^w - x_i}(a_i) = z_i$.

Example 9.10 Suppose that $k = 9$ (and hence $\ell = 3$). There are three hash chains:

$$y_1^0 \to y_1^1 \to y_1^2 \to y_1^3 \to y_1^4 \to y_1^5 \to y_1^6 \to y_1^7 \to z_1$$

$$y_2^0 \to y_2^1 \to y_2^2 \to y_2^3 \to y_2^4 \to y_2^5 \to y_2^6 \to y_2^7 \to z_2$$

$$y_3^0 \to y_3^1 \to y_3^2 \to y_3^3 \to y_3^4 \to y_3^5 \to y_3^6 \to y_3^7 \to z_3.$$

Therefore, the public key is (z_1, z_2, z_3). Now suppose we want to sign the message 011101001. We have $x_1 = 011 = 3$, $x_2 = 101 = 5$, and $x_3 = 001 = 1$. So we release the values $a_1 = y_1^3$, $a_2 = y_2^5$, and $a_3 = y_3^1$: