It should be emphasized that even 15 years of lead time to develop post-quantum cryptographic algorithms leaves little margin for delay. The development, standardization, and deployment of new cryptography technologies can easily take 20 years or more. Thus, post-quantum cryptography is viewed by many as a serious problem of immediate and pressing concern. In particular, NIST is giving a high priority to quantum cryptography, sponsoring the first standardization conference for post-quantum cryptography in 2018. It is also worth noting that, in 2015, the NSA announced its intention to transition to post-quantum cryptography.

9.2 Lattice-based Cryptography

Lattice-based cryptography has been of interest for over twenty years. We first describe the *NTRU* public-key cryptosystem. Then, after a discussion of the basic theory of lattices, we give a brief introduction to cryptography whose security rests on the presumed difficulty of the **Learning With Errors** problem.

9.2.1 NTRU

NTRU is a public-key cryptosystem, due to Hoffstein, Pipher, and Silverman, that was introduced at the CRYPTO '96 rump session. The current version of *NTRU* is known as *NTRUEncrypt*. It is a very fast cryptosystem that is easy to implement. It is also of interest because its security is based on certain lattice problems and thus it is considered to be a practical example of post-quantum cryptography. (We will discuss these lattice problems in the next section.)

NTRUEncrypt is defined in terms of three parameters, N, p, and q, which are fixed integers. Computations are performed in the ring $\mathcal{R} = \mathbb{Z}[x]/(x^N - 1)$. Multiplication of two polynomials is easy in $\mathcal{R}$; it suffices to compute the product of two polynomials in $\mathbb{Z}[x]$ and then reduce all exponents modulo N.

For example, suppose $N = 3$ and we want to compute the product $(x^2 + 3x + 1)(2x^2 + x - 4)$ in $\mathcal{R}$. We compute as follows:

$$\begin{aligned}(x^2+3x+1)(2x^2+x-4) &= 2x^4+7x^3+x^2-11x-4\\ &= 2x+7+x^2-11x-4\\ &= x^2-9x+3.\end{aligned}$$

It is often convenient to represent a polynomial in $\mathcal{R}$ by its vector of coefficients:

$$a(x) = \sum_{i=0}^{N-1} a_i x^i \text{ corresponds to } \mathbf{a} = (a_0, a_1, \dots, a_{N-1}).$$

Suppose we have

$$a(x) = \sum_{i=0}^{N-1} a_i x^i,$$

$$b(x) = \sum_{i=0}^{N-1} b_i x^i,$$

and

$$c(x) = a(x)b(x) = \sum_{i=0}^{N-1} c_i x^i.$$

The corresponding coefficient vectors have the relation

$$\mathbf{c} = \mathbf{a} \star \mathbf{b},$$

where "$\star$" is a ***convolution operation***. Specifically, for $0 \leq i \leq N-1$, we have that

$$c_i = \sum_{j=0}^{N-1} a_j b_{i-j}, \tag{9.1}$$

where all subscripts are reduced modulo N.

In our description of *NTRUEncrypt*, we will sometimes use the notation of coefficient vectors and the convolution operation. But of course this is exactly the same thing as multiplication in $\mathcal{R}$.

At various points in the *NTRUEncrypt* encryption and decryption process, coefficients will be reduced modulo p or modulo q. These parameters have the following properties: q will be quite a bit larger than p, and q and p should be relatively prime. Also, p should be odd. The values $p = 3$ and $q = 2048$ are popular choices. Finally, N is usually taken to be a prime; $N = 401$ is a currently recommended value.

Various operations in *NTRUEncrypt* require certain "centered" modular reductions, which we define now.

Definition 9.1: For an odd integer n and integers a and b, define

$$a \textsf{ mods } n = b \text{ if } a \equiv b \pmod{n} \text{ and } -\tfrac{n-1}{2} \leq b \leq \tfrac{n}{2},$$

For example $a \textsf{ mods } 5 \in \{-2,-1,0,1,2\}$, whereas $a \bmod 5 \in \{0,1,2,3,4\}$.

We now describe the public and private keys used in *NTRUEncrypt*. First, $F(x)$ and $G(x)$ are secret polynomials chosen from $\mathcal{R}$. All coefficients of $F(x)$ and $G(x)$ will be in the set $\{-1,0,1\}$. Next, define $f(x) = 1 + pF(x)$ and $g(x) = pG(x)$. Finally, compute $f^{-1}(x)$ in the ring $\mathcal{R} \textsf{ mods } q$, and then compute $h(x) = f^{-1}(x)g(x) \textsf{ mods } q$. After this is done, F and G can be discarded.

The public key is the coefficient vector $\mathbf{h}$ and the private key is the coefficient vector $\mathbf{f}$. The polynomial $g(x)$ is used in the construction of the public key $h(x)$; $g(x)$ is not part of the public or private key, but it should be kept secret and then discarded after $h(x)$ is formed.

The polynomial $f^{-1}(x)$ can be computed using the EXTENDED EUCLIDEAN ALGORITHM for polynomials. Let $c(x) = \gcd(f(x), x^N - 1)$, which is computed in $\mathbb{Z}_q[x]$. The extended Euclidean algorithm computes polynomials $a(x), b(x) \in \mathbb{Z}_q[x]$ such that

$$a(x)f(x) + b(x)(x^N - 1) = c(x).$$

Then $f^{-1}(x)$ exists if and only if $c(x) = 1$. Further, if $c(x) = 1$, then $f^{-1}(x) = a(x) \text{ mods } q$.

A plaintext $\mathbf{m}$ is an N-tuple in the set $\{-1, 0, 1\}^N$. The encryption operation in *NTRUEncrypt* is randomized. First, $\mathbf{r} \in \{-1, 0, 1\}^N$ is chosen uniformly at random from a specified subset of $\mathcal{R}$. The ciphertext $\mathbf{y}$ is computed as

$$\mathbf{y} = \mathbf{r} \star \mathbf{h} + \mathbf{m} \text{ mods } q.$$

To decrypt a ciphertext $\mathbf{y}$, perform the following operations:

1. Compute $\mathbf{a} = \mathbf{f} \star \mathbf{y} \text{ mods } q$.
2. Compute $\mathbf{m}' = \mathbf{a} \text{ mods } p$.

If all goes well, it will be the case that $\mathbf{m}' = \mathbf{m}$.

First, it is easy to verify that $\mathbf{a} \equiv \mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m} \pmod q$. This is done as follows:

$$\begin{aligned} \mathbf{a} &\equiv \mathbf{f} \star \mathbf{y} \pmod q \\ &\equiv \mathbf{f} \star (\mathbf{r} \star \mathbf{h} + \mathbf{m}) \pmod q \\ &\equiv \mathbf{f} \star (\mathbf{r} \star \mathbf{f}^{-1} \star \mathbf{g} + \mathbf{m}) \pmod q \\ &\equiv \mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m} \pmod q. \end{aligned}$$

Now, suppose that this congruence is actually an equality in $\mathcal{R}$, i.e.,

$$\mathbf{a} = \mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m}. \tag{9.2}$$

This happens if and only if every coefficient of $\mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m}$ lies in the interval

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

which will hold with high probability if the parameters of the system are chosen in a suitable way.

Now, assuming that (9.2) holds and reducing modulo p, we have

$$\begin{aligned} \mathbf{a} &\equiv \mathbf{r} \star \mathbf{g} + \mathbf{f} \star \mathbf{m} \pmod p \\ &\equiv \mathbf{r} \star p\,\mathbf{G} + (1 + p\,\mathbf{F}) \star \mathbf{m} \pmod p \\ &\equiv \mathbf{m} \pmod p. \end{aligned}$$

From this relation, we see that

$$\mathbf{m} = \mathbf{a} \text{ mods } p,$$

because all of the coefficients of $\mathbf{m}$ are in the set $\{-1, 0, 1\}$. Therefore the ciphertext is decrypted correctly.

A concise description of *NTRUEncrypt* is presented as Cryptosystem 9.1. We illustrate the encryption and decryption processes with an example.

Cryptosystem 9.1: *NTRUEncrypt*

Suppose p, q, and N are integers, where $q \gg p$, q and p are relatively prime, p is odd, and N is prime. Typical values for these parameters are $p = 3$, $q = 2048$, and $N = 401$.

Let $\mathcal{P} = \{-1, 0, 1\}^N$ and $\mathcal{C} = (\mathbb{Z}_q)^N$. Choose $\mathbf{F}, \mathbf{G} \in \{-1, 0, 1\}^N$, let $\mathbf{f} = 1 + p\,\mathbf{F}$, let $\mathbf{g} = p\,\mathbf{G}$, and define $\mathbf{h} = \mathbf{f}^{-1}\mathbf{g} \text{ mods } q$. The associated key is $K = (\mathbf{f}, \mathbf{h})$, where $\mathbf{f}$ is private and $\mathbf{h}$ is public.

Now define

$$e_K(\mathbf{m}) = \mathbf{y} = \mathbf{r} \star \mathbf{h} + \mathbf{m} \text{ mods } q$$

for a randomly chosen $\mathbf{r} \in \{-1, 0, 1\}^N$, and

$$d_K(\mathbf{y}) = (\mathbf{f} \star \mathbf{y} \text{ mods } q) \text{ mods } p.$$

Example 9.1 Suppose we take $N = 23$, $p = 3$ and $q = 31$. Let

$$F(x) = x^{18} - x^9 + x^8 - x^4 - x^2, \text{ so } f(x) = 3x^{18} - 3x^9 + 3x^8 - 3x^4 - 3x^2 + 1$$

and

$$G(x) = x^{17} + x^{12} + x^9 + x^3 - x, \text{ so } g(x) = 3x^{17} + 3x^{12} + 3x^9 + 3x^3 - 3x.$$

Next we compute

$$\begin{aligned} h(x) = {} & -13x^{22} - 15x^{21} + 12x^{19} - 14x^{18} + 8x^{16} - 14x^{15} - 6x^{14} + 14x^{13} \\ & - 3x^{12} + 7x^{11} - 5x^{10} - 14x^9 + 3x^8 + 10x^7 + 5x^6 - 8x^5 + 4x^2 + x + 8. \end{aligned}$$

Suppose we wish to encrypt the plaintext

$$m(x) = x^{15} - x^{12} + x^7 - 1,$$

and we choose

$$r(x) = x^{19} + x^{10} + x^6 - x^2.$$

The ciphertext is

$$\begin{aligned} y(x) = {} & 5x^{22} - 15x^{21} + 4x^{20} + 8x^{19} + 10x^{18} - 15x^{17} + 6x^{16} + 8x^{15} - 8x^{14} \\ & + 3x^{13} - 10x^{12} - 7x^{11} - x^{10} - 9x^9 + 12x^8 - 14x^7 + 15x^6 - 10x^5 \\ & + 15x^4 - 14x^3 - 5x^2 - 15x - 3. \end{aligned}$$

The decryption process will begin by computing

$$\begin{aligned} a(x) = {} & 6x^{22} + 3x^{21} - 6x^{20} - 3x^{19} - 3x^{17} + 7x^{15} + 6x^{13} - x^{12} - 9x^{11} + 3x^{10} \\ & + 3x^9 - 5x^7 + 6x^4 + 3x^3 + 6x^2 - 3x + 5. \end{aligned}$$

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 **F**, **G**, **r**, and **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 combinations of basis vectors. If there are r vectors in the basis, then we have an ***r-dimensional vector space***. Restating this using mathematical notation, suppose the r basis vectors are $\mathbf{b}^1, \dots, \mathbf{b}^r$. The vector space generated by this basis consists of all the vectors of the form

$$\alpha_1 \mathbf{b}^1 + \cdots + \alpha_r \mathbf{b}^r,$$

where $\alpha_1, \dots, \alpha_r$ are arbitrary real numbers.

Now, a lattice is very similar, except that the vectors in the lattice are ***integer linear combinations*** of basis vectors. That is, the lattice generated by the basis consists of all the vectors of the form

$$\alpha_1 \mathbf{b}^1 + \cdots + \alpha_r \mathbf{b}^r,$$

where $\alpha_1, \dots, \alpha_r$ are arbitrary integers.

For a vector $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{R}^n$, we define the ***norm*** of $\mathbf{v}$, which is denoted by $\|\mathbf{v}\|$, as follows:

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} {v_i}^2}.$$

Two fundamental problems in the setting of lattices are the **Shortest Vector** problem and the **Closest Vector** problem. We define these as Problems 9.1 and 9.2. In the specification of these problems, an "instance" consists of a lattice. However, we will always consider a lattice to be specified or represented by giving a basis for the lattice; this is the phraseology used in these problems.

Problem 9.1: Shortest Vector

Instance: A basis for a lattice $\mathcal{L}$ in $\mathbb{R}^n$.
Find: A vector $\mathbf{v} \in \mathcal{L}$, $\mathbf{v} \neq (0, \dots, 0)$, such that $\|\mathbf{v}\|$ is minimized. Such a vector $\mathbf{v}$ is called a ***shortest vector*** in $\mathcal{L}$.

Problem 9.2: Closest Vector

Instance: A basis for a lattice $\mathcal{L}$ in $\mathbb{R}^n$ and a vector $\mathbf{w} \in \mathbb{R}^n$ that is not in $\mathcal{L}$.
Find: A vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v} - \mathbf{w}\|$ is minimized. Such a vector $\mathbf{v}$ is called a ***closest vector*** to $\mathbf{w}$ in $\mathcal{L}$.

One way in which an adversary could break *NTRUEncrypt* would be to compute the polynomials $f(x)$ and $g(x)$ that were used to construct the public key $\mathbf{h}$. Denote $\mathbf{h} = (h_0, \dots, h_{N-1})$ and consider the lattice $\mathcal{L}_{\mathbf{h}}$ whose basis consists of the rows of the following $2N$ by $2N$ matrix:

$$M = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right)$$

The lattice $\mathcal{L}_{\mathbf{h}}$ consists of the following vectors:

$$\mathcal{L}_{\mathbf{h}} = \{(\mathbf{a}, \mathbf{b}) \in \mathbb{Z}^{2N} : \mathbf{a} \star \mathbf{h} = \mathbf{b}\}.$$

From the way in which $\mathbf{h}$ is constructed, we have that

$$\mathbf{f} \star \mathbf{h} \equiv \mathbf{g} \bmod q,$$

where $\mathbf{f}$ and $\mathbf{g}$ are the coefficient vectors of $f(x)$ and $g(x)$, respectively. This means that

$$\mathbf{f} \star \mathbf{h} - \mathbf{g} = q\,\mathbf{t}$$

for some integer vector $\mathbf{t}$. It is then straightforward to compute

$$(\mathbf{f}, -\mathbf{t})M = (\mathbf{f}, \mathbf{g}),$$

so $(\mathbf{f}, \mathbf{g}) \in \mathcal{L}_{\mathbf{h}}$.

Further, the vector $(\mathbf{f}, \mathbf{g})$ has a small norm, since all of its coefficients are in the set $\{-p, -1, 0, 1, p\}$. More precisely, $(\mathbf{f}, \mathbf{g})$ has roughly $N/3$ coefficients equal to each of $-p$, -1, 1, and p, and the remaining $2N/3$ coefficients are equal to 0. So the norm of $(\mathbf{f}, \mathbf{g})$ is approximately

$$\sqrt{\frac{2N(1+p^2)}{3}} = 2\sqrt{\frac{5N}{3}}.$$

However, a vector of length $2N$ whose co-ordinates take on random values in $[-q/2, q/2]$ would (on average) have norm approximately equal to

$$q\sqrt{\frac{N}{6}},$$

which is much larger (recall that we are assuming $p = 3$ and $q = 2048$).

It therefore seems plausible that $(\mathbf{f}, \mathbf{g})$ is the shortest vector in the lattice $\mathcal{L}_{\mathbf{h}}$. Since solving the **Shortest Vector** problem is believed to be difficult, it should not be possible for the adversary to find the private key $\mathbf{f}$.

An adversary might also attempt to decrypt a specific ciphertext. It turns out that this can be modeled as an instance of the **Closest Vector** problem. The vector

$(0, \mathbf{y})$ is in fact quite close to the vector $(\mathbf{r}, \mathbf{r} \star \mathbf{h} \bmod s\ q)$, which is in the lattice $\mathcal{L}_{\mathbf{h}}$. More precisely,

$$(0, \mathbf{y}) = (\mathbf{r}, \mathbf{r} \star \mathbf{h} \ \mathsf{mods}\ q) + (-\mathbf{r}, \mathbf{m}),$$

and $(-\mathbf{r}, \mathbf{m})$ has small norm. Therefore, it seems reasonable that the closest vector to $(0, \mathbf{y})$ is $(\mathbf{r}, \mathbf{r} \star \mathbf{h} \ \mathsf{mods}\ q)$. Solving the **Closest Vector** problem reveals the vector $\mathbf{r}$, which allows $\mathbf{y}$ to be decrypted.

It is important to emphasize that there is no proof that breaking *NTRUEncrypt* is as hard as solving the **Shortest Vector** problem or the **Closest Vector** problem. Thus, *NTRUEncrypt* cannot (currently, at least) be regarded as a provably secure cryptosystem. We give an example of a provably secure lattice-based public-key cryptosystem in the next subsection.

9.2.3 Learning With Errors

Given a prime q, it is possible to find solutions to a system of linear equations in n variables over $\mathbb{Z}_q$ efficiently. However, by carefully introducing randomness into the system we obtain a new problem, known as the **Learning With Errors** (or **LWE**) problem, that is believed to be difficult to solve.

Problem 9.3: Learning With Errors

Instance: A prime q, an integer n, a discrete random variable $\mathbf{E}$ with probability distribution χ defined on the set $\mathbb{Z}_q$ and m samples $(\mathbf{a^i}, b^i) \in (\mathbb{Z}_q)^{n+1}$. The m samples are all constructed from a secret $\mathbf{s} = (s_1, s_2, \dots, s_n) \in (\mathbb{Z}_q)^n$. For $1 \leq i \leq m$, $\mathbf{a^i} = (a^i_1, \dots, a^i_n)$ is chosen uniformly at random from $(\mathbb{Z}_q)^n$, e^i is chosen using the probability distribution χ and

$$b^i = e^i + \sum_{j=1}^{n} a^i_j s_j \bmod q.$$

Find: The secret $(s_1, s_2, \dots, s_n)$.

Informally, **LWE** can be regarded as the problem of finding a solution modulo q to the approximate system of linear equations

$$\begin{aligned} a^1_1 x_1 + a^1_2 x_2 + \cdots + a^1_n x_n &\approx b^1, \\ a^2_1 x_1 + a^2_2 x_2 + \cdots + a^2_n x_n &\approx b^2, \\ &\vdots \\ a^m_1 x_1 + a^m_2 x_2 + \cdots + a^m_n x_n &\approx b^m. \end{aligned}$$

The solution is unique if there are enough equations in the system. Constructing an **LWE** system that is difficult to solve requires a value of q that is significantly larger than n as well as a careful choice of probability distribution χ for the random variable **E**. Most proposals use a distribution in which the probability of a given error e increases the closer e is to 0 (where closeness is defined by treating e as an integer in the range $-\lfloor q/2 \rfloor$ to $\lfloor q/2 \rfloor$), with 0 being the most likely error. If the variance of the distribution is too high then the errors risk obscuring the rest of the information in the system. However, if it is too small (for example, if **E** is uniformly zero) then the corresponding **LWE** problem will not be secure.

Existing literature has used very sophisticated techniques to show that particular families of distributions are suitable for use in cryptographic constructions based on **LWE**. The **LWE** problem is of interest to cryptographers because the average-case difficulty of solving instances of **LWE** can be shown to be based on the worst-case difficulty of solving a certain lattice problem that is believed to be hard. The particular lattice problem used in the reduction is a decision problem known as the **Gap Shortest Vector** problem, which (naturally) is closely related to the **Shortest Vector** problem. There is no known efficient quantum algorithm to solve this problem, so cryptographic systems based on **LWE** are regarded as post-quantum and fall under the general category of lattice-based cryptography.

Cryptosystem 9.2 (the *Regev Cryptosystem*) is an example of a cryptosystem based on **LWE** that can be used to encrypt a single bit. We note that the ciphertext involves a sum of samples, and that the "errors" in the samples are also summed. It follows that in order for decryption to be possible, it is necessary to choose the distribution χ in such a way that the overall error in the ciphertext is not so great as to obscure the distinction between values close to 0 and values close to $q/2$. It is possible to show that Cryptosystem 9.2 is secure against chosen plaintext attacks provided that

- samples constructed from a secret **s** and errors following the probability distribution χ cannot be distinguished from uniformly distributed elements of $(\mathbb{Z}_q)^{n+1}$, and
- an algorithm for distinguishing these LWE samples from uniform elements can be used to break **LWE**.

Example 9.2 Let $n = 3$ and $q = 11$. Suppose that **E** is the discrete random variable that takes on each of the values 0, 1, or -1 with probability $1/3$. Suppose the secret key is $(1, 2, 3)$, and the public key consists of the three samples $((5, 8, 10), 7)$, $((4, 9, 1), 4)$, and $((3, 6, 0), 3)$.

To encrypt the message $t = 1$ we choose a random subset of $\{1, 2, 3\}$, say $S = \{1, 3\}$. Then the ciphertext is $(\mathbf{a}, b)$ where

$$\mathbf{a} = (5, 8, 10) + (3, 6, 0) = (8, 3, 10)$$

and

$$b = 7 + 3 + \lfloor 11/2 \rfloor = 4.$$

To decrypt the ciphertext $((8, 3, 10), 4)$ we compute

$$4 - 8 \times 1 - 3 \times 2 - 10 \times 3 = 4.$$

Because 4 is closer to 5 than to 0, the decrypted message is 1, as expected. □

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 $b^i = \mathbf{E} + \sum_{j=1}^{n} a^i_j s_j$.

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 $\mathbf{x} = (x_1, \dots, x_n)$ and $\mathbf{y} = (y_1, \dots, y_n)$. 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 $\mathbf{x} = (x_1, \dots, x_n)$ and $\mathbf{y} = (y_1, \dots, y_n)$, 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

$2, \dots, \lfloor (d-1)/2 \rfloor$. 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

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

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 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**

Problem 9.4: Multivariate Quadratic Equations

Instance: A finite field $\mathbb{F}_q$ and a system of m quadratic equations in n variables:

$$\begin{array}{rcl} f_1(x_1, x_2, \ldots, x_n) & = & d_1, \\ f_2(x_1, x_2, \ldots, x_n) & = & d_2, \\ \vdots & \vdots & \vdots \\ f_m(x_1, x_2, \ldots, x_n) & = & d_m, \end{array} \tag{9.3}$$

where, for all k with $1 \leq k \leq m$, the polynomial f_k has the form

$$f_k(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \sum_{j=i}^{n} a_{ij} x_i x_j + \sum_{i=1}^{n} b_i x_i + c,$$

with a_{ij}, b_i, and c chosen uniformly at random from $\mathbb{F}_q$, for all i, j with $1 \leq i, j \leq n$.

Question: Find a vector $(s_1, s_2, \ldots, s_n) \in (\mathbb{F}_q)^n$ that satisfies the equations $f_i(s_1, s_2, \ldots, s_n) = d_i$ for all i with $1 \leq i \leq m$.

Equations problem, and is usually abbreviated as the **MQ** problem. We present it as Problem 9.4.

The **MQ** problem is NP-hard over any finite field. It is used as the basis of both public-key cryptosystems and signature scheme, which we discuss in the next two subsections.

9.4.1 Hidden Field Equations

The design of cryptosystems based on the **MQ** problem follows a similar strategy to that of the *McEliece Cryptosystem*. Namely, it involves starting with a special case of the problem that is easy to solve, and then disguising it with the aim of making it appear like a general instance of the problem. In order to explore an example of this approach, we consider the cryptosystem known as *Hidden Field Equations* (*HFE*), proposed in 1996 by Jacques Patarin. In *HFE*, the special system of multivariate quadratic equations is constructed from a univariate polynomial over an extension $\mathbb{F}_{q^n}$ of the field $\mathbb{F}_q$. The following example shows how this can be done.

Example 9.4 In Example 7.7, we saw that the field $\mathbb{F}_8 = \mathbb{F}_{2^3}$ is an extension of the field $\mathbb{F}_2$. All of its elements can be written in the form $a\theta^2 + b\theta + c$ with $a, b, c \in \mathbb{F}_2$, where θ satisfies $\theta^3 + \theta + 1 = 0$ (note that we have switched to using θ in place of x here to avoid confusion with the other notation used in this example). As discussed in Example 7.7, there is a one-to-one correspondence between elements of $(\mathbb{F}_2)^3$ and $\mathbb{F}_8$ where (a, b, c) corresponds to $a\theta^2 + b\theta + c$.

Consider the polynomial $f(X) = X^5 + X^2 + 1$ with coefficients from $\mathbb{F}_8$. Any solution to the equation

$$f(X) \;=\; 0 \tag{9.4}$$

in $\mathbb{F}_8$ can be written in the form $x_1\theta^2 + x_2\theta + x_3$. Substituting this expression in place of X gives

$$\begin{aligned} f(X) &= X^5 + X^2 + 1, \\ &= (x_1\theta^2 + x_2\theta + x_3)^5 + (x_1\theta^2 + x_2\theta + x_3)^2 + 1. \end{aligned}$$

Now, if we expand the terms in parentheses, we will obtain an expression involving powers of θ up to θ^{10}. However, as in Example 7.7, each of these powers of θ can be expressed in the form $a\theta^2 + b\theta + c$ for suitable $a, b, c \in \mathbb{F}_2$. Once we express the powers of θ in this form and then collect together like powers of θ, we obtain

$$f(x_1\theta^2 + x_2\theta + x_3) = \theta^2(x_3x_1 + x_3x_2 + x_1) + \theta(x_3x_1 + x_2) + x_2x_1 + x_2 + x_1 + 1.$$

(We will discuss how this simplification can be carried out conveniently in Examples 9.5 and 9.6.) If we compare the coefficients of the various powers of θ, we can observe that a solution $a\theta^2 + b\theta + c \in \mathbb{F}_8$ to (9.4) corresponds to a solution $(a, b, c) \in (\mathbb{F}_2)^3$ to the following system of multivariate quadratic equations:

$$\begin{aligned} g_1(\mathbf{x}) &= x_3x_1 + x_3x_2 + x_1 &&= 0 \\ g_2(\mathbf{x}) &= x_3x_1 + x_2 &&= 0 \\ g_3(\mathbf{x}) &= x_2x_1 + x_2 + x_1 + 1 &&= 0. \end{aligned} \tag{9.5}$$

There exist efficient algorithms for finding solutions of systems of univariate polynomial equations over finite fields. Hence, applying these techniques to (9.4) over $\mathbb{F}_8$ gives an efficient way to find solutions to the system of multivariate quadratic equations (9.5) over $\mathbb{F}_2$. □

Any univariate polynomial over an extension field $\mathbb{F}_{q^n}$ can be turned into a system of multivariate polynomial equations over the field $\mathbb{F}_q$ by following the approach illustrated in Example 9.4. However, in general, these equations will have degree greater than two. To ensure we obtain a system of quadratic equations, we have to choose our univariate polynomial carefully. The following examples show how this can be done.

Example 9.5 Consider the term X^2 that appears in the polynomial $f(X)$ of Example 9.4. We observe that, if a and b are elements of $\mathbb{F}_8$, then

$$(a + b)^2 = a^2 + ab + ab + b^2 = a^2 + b^2,$$

since $\mathbb{F}_8$ has characteristic 2. This means that, when we substitute $x_1\theta^2 + x_2\theta + x_3$ in place of X, we see that

$$\begin{aligned} X^2 &= (x_1\theta^2 + x_2\theta + x_3)^2, \\ &= (x_1\theta^2)^2 + (x_2\theta)^2 + {x_3}^2, \\ &= {x_1}^2\theta^4 + {x_2}^2\theta^2 + {x_3}^2. \end{aligned}$$

However, we observe that x_3, x_2, and x_1 all represent elements of $\mathbb{F}_2$ and hence they can take on only the values 0 or 1. Now $0^2 = 0$ and $1^2 = 1$, so we can conclude that ${x_1}^2 = x_1$, ${x_2}^2 = x_2$, and ${x_3}^2 = x_3$, so the above expression becomes

$$\begin{aligned} x_1\theta^4 + x_2\theta^2 + {x_3}^2 &= x_1(\theta^2 + \theta) + x_2\theta^2 + x_3, \\ &= \theta^2(x_1 + x_2) + \theta x_1 + x_3. \end{aligned}$$

Thus we see that X^2 translates into linear equations in the variables x_3, x_2 and x_1. Similarly, we can deduce that X^{2^i} will also give rise to linear equations for any $i \geq 0$. □

Example 9.6 Now consider the term X^5 that appears in $f(X)$. We note that

$$X^5 = (X^4)X = X^{2^2}X^{2^0}.$$

Hence we see that X^5 can be written as the product of two terms that are linear in the variables x_3, x_2 and x_1, and so the resulting expression will contain terms that are quadratic in these variables:

$$\begin{aligned} X^5 &= X^4X \\ &= (x_1\theta^2 + x_2\theta + x_3)^4(x_1\theta^2 + x_2\theta + x_3), \\ &= (x_1\theta^8 + x_2\theta^4 + x_3)(x_1\theta^2 + x_2\theta + x_3), \\ &= x_1\theta^{10} + x_2x_1\theta^9 + x_3x_1\theta^8 + x_2x_1\theta^6 + x_2\theta^5 + x_3x_2\theta^4 + x_3x_1\theta^2 + x_3x_2\theta + x_3, \\ &= x_1(\theta + 1) + x_2x_1\theta^2 + x_3x_1\theta + x_2x_1(\theta^2 + 1) + x_2(\theta^2 + \theta + 1) \\ &\quad + x_3x_2(\theta^2 + \theta) + x_3x_1\theta^2 + x_3x_2\theta + x_3, \\ &= (x_2 + x_3x_2 + x_3x_1)\theta^2 + (x_1 + x_3x_1 + x_2)\theta + x_1 + x_2x_1 + x_2 + x_3. \end{aligned}$$

Similarly, any term of the form $X^{2^i+2^j}$ with $i, j \geq 0$ will give rise to terms that have degree at most two in the variables x_i. □

More generally, we can extend the arguments used in Examples 9.5 and 9.6 to the case of the field $\mathbb{F}_q$ to show that choosing a univariate polynomial over $\mathbb{F}_{q^n}$ of the form

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}X^{q^i+q^j} + \sum_{i=0}^{n-1} b_iX^{q^i} + c$$

ensures that, when we translate this polynomial into a system of multivariate polynomials over $\mathbb{F}_q$, the degree of these polynomials is at most two.

Cryptosystem 9.4: *Hidden Field Equations*

Let $\mathbb{F}_q$ be a finite field and let $n > 0$ be an integer. The private key consists of invertible affine transformations R and S, together with a univariate polynomial $f(X)$ with coefficients from $\mathbb{F}_{q^n}$ that has the form

$$f(X) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} X^{q^i+q^j} + \sum_{i=0}^{n-1} b_i X^{q^i} + c.$$

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and, for i from 1 to n, let $g_i(\mathbf{x})$ be the polynomials obtained by representing $f(X)$ as a system of n quadratic polynomials in n variables over $\mathbb{F}_q$. The public key consists of the system of n quadratic polynomials in n variables over $\mathbb{F}_q$ that are given by

$$\begin{pmatrix} g_1^{\mathsf{pub}}(\mathbf{x}) \\ g_2^{\mathsf{pub}}(\mathbf{x}) \\ \vdots \\ g_n^{\mathsf{pub}}(\mathbf{x}) \end{pmatrix} = R \begin{pmatrix} g_1(S(\mathbf{x})) \\ g_2(S(\mathbf{x})) \\ \vdots \\ g_n(S(\mathbf{x})) \end{pmatrix}.$$

The plaintexts are elements of $(\mathbb{F}_q)^n$, and a plaintext $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ is encrypted by computing $\mathbf{y} = (g_1^{\mathsf{pub}}(\mathbf{a}), g_2^{\mathsf{pub}}(\mathbf{a}), \ldots, g_n^{\mathsf{pub}}(\mathbf{a}))$.

A ciphertext $\mathbf{y}$ is decrypted through the following steps:

1. Compute $\mathbf{y}' = R^{-1}(\mathbf{y})$.
2. Find all solutions $\mathbf{z} \in \mathbb{F}_{q^n}$ to the equation $f(X) = \mathbf{y}'$ (here $\mathbf{y}'$ is interpreted to be an element of $\mathbb{F}_{q^n}$).
3. Calculate $S^{-1}(\mathbf{z})$ for all solutions found in the previous step. One of these solutions is the desired plaintext, $\mathbf{a}$. (By using some redundancy when representing plaintexts as elements of $(\mathbb{F}_q)^n$, it is possible to ensure that the correct solution can be identified at this point.)

Now that we have a way to construct a system of equations for which we know how to find solutions, it is necessary to "hide" the fact that they were obtained from a polynomial over an extension field in this way. This is done by changing variables through the use of affine transformations that map an n-tuple $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ to the element $M\mathbf{x}^T + \mathbf{v}^T$, where M is an invertible $n \times n$ matrix with entries from $\mathbb{F}_q$, the vector $\mathbf{v} \in (\mathbb{F}_q)^n$ and the superscript "T" denotes "transpose." A full description of *HFE* is presented as Cryptosystem 9.4. We now give a toy example to show how encryption and decryption work.

Example 9.7 Suppose we continue to work with the extension $\mathbb{F}_8$ of the field $\mathbb{F}_2$ constructed in Example 7.7. Take $f(X) = X^3 \in \mathbb{F}_8[X]$ and define S and R to be the following linear transformations:

$$S : \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_2 + 1 \\ x_1 \\ x_3 + 1 \end{pmatrix}$$

$$R : \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} x_2 + 1 \\ x_3 \\ x_1 \end{pmatrix}.$$

We then have

$$\begin{aligned} g_1(\mathbf{x}) &= x_2x_3 + x_1, \\ g_2(\mathbf{x}) &= x_1x_3 + x_2x_3 + x_2, \\ g_3(\mathbf{x}) &= x_1x_2 + x_1 + x_2 + x_3. \end{aligned}$$

It then follows that

$$g_1(S(\mathbf{x})) = x_1(x_3 + 1) + (x_2 + 1) = x_1x_3 + x_1 + x_2 + 1,$$

and, similarly, we can determine that

$$g_2(S(\mathbf{x})) = x_1x_3 + x_2x_3 + x_1 + x_3 + 1$$

and

$$g_3(S(\mathbf{x})) = x_1x_2 + x_2 + x_3.$$

Thus we have

$$\begin{aligned} g_1^{\mathsf{pub}}(\mathbf{x}) &= x_1x_3 + x_2x_3 + x_1 + x_3, \\ g_2^{\mathsf{pub}}(\mathbf{x}) &= x_1x_2 + x_2 + x_3, \\ g_3^{\mathsf{pub}}(\mathbf{x}) &= x_1x_3 + x_1 + x_2 + 1. \end{aligned}$$

We can encrypt the plaintext $\mathbf{a} = (1, 1, 0)$ by evaluating g_1^{pub}, g_2^{pub}, and g_3^{pub} at $\mathbf{a}$, which results in the ciphertext $\mathbf{y} = (1, 0, 1)$.

Now the inverse of the transformation R sends $(x_1, x_2, x_3)^T$ to the value $(x_3, x_1 + 1, x_2)^T$. Hence $R^{-1}(\mathbf{y}^T) = (1, 0, 0)^T$, which corresponds to the element θ^2 in $\mathbb{F}_8$.

The next step in decryption is to find the solutions to the equation $X^3 = \theta^2$; over $\mathbb{F}_8$ there is a unique solution $X = \theta + 1$, which corresponds to $(0, 1, 1)$. The inverse of S sends $(x_1, x_2, x_3)^T$ to $(x_2, x_1 + 1, x_3 + 1)^T$. Hence $S^{-1}((0, 1, 1)^T) = (1, 1, 0)^T$, and we have recovered our original plaintext. $\square$

Experiments involving ***Gröbner basis algorithms*** (a class of algorithms that include some of the fastest known techniques for solving general instances of the

MQ problem) indicate that solving systems of equations arising from *HFE* may be significantly easier than solving randomly generated systems of equations. This suggests that the use of affine transformations is not entirely effective in hiding the specialized structure of these systems of equations. Many variations on *HFE* have been proposed in order to strengthen the scheme, including omitting some of the multivariate equations from the system or adding random quadratic equations to the equations in the system. Although *HFE* itself is no longer regarded as secure for any practical parameter sizes, the underlying ideas continue to inspire the design of new multivariate cryptosystems.

9.4.2 The Oil and Vinegar Signature Scheme

The *Oil and Vinegar Signature Scheme*, proposed by Jacques Patarin in 1997, is an example of a multivariate signature scheme. As in the case of *HFE* (which was discussed in Section 9.4.1), it involves a system of multivariate quadratic equations that is easy to solve, disguised by the use of an affine transformation. In this case, the initial system consists of n polynomial equations in $2n$ variables $x_1, x_2, \ldots, x_{2n}$ over a finite field $\mathbb{F}_q$. The first n variables $x_1, x_2, \ldots, x_n$ are referred to as the ***vinegar variables*** and the remaining variables $x_{n+1}, x_{n+2}, \ldots, x_{2n}$ are the ***oil variables***. These names reflect the fact that, when oil and vinegar are combined to make a salad dressing, they are initially separated into distinct layers, and then they are shaken up to mix them. For this scheme, the oil and vinegar variables are "separated" in the quadratic polynomials used in the signing key, but are "mixed" by the application of an affine transformation in order to construct the verification key.

Specifically, the n quadratic polynomials that make up the signing key for the *Oil and Vinegar Signature Scheme* have the form

$$f_k(x_1, x_2, \ldots, x_{2n}) = \sum_{i=1}^{2n} \sum_{j=1}^{n} a_{ij}^k x_i x_j + \sum_{i=1}^{2n} b_i^k x_i + c^k \tag{9.6}$$

for all k with $1 \leq k \leq n$. What is special about these equations is that there are terms involving the product of two vinegar variables, or one vinegar variable and one oil variable, but there are no terms involving the product of two oil variables. Given a vector $(m_1, m_2, \ldots, m_n) \in (\mathbb{F}_q)^n$, we can easily exploit this structure to find a solution to the following system of multivariate quadratic equations:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_{2n}) &= m_1, \\ f_2(x_1, x_2, \ldots, x_{2n}) &= m_2, \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_{2n}) &= m_n. \end{aligned} \tag{9.7}$$

To do this, we choose random values $v_1, v_2, \ldots, v_n \in \mathbb{F}_q$ for the vinegar variables. When these values are substituted in (9.7), we are left with a system of n linear equations in the n oil variables, which we can then solve to find a solution

$(v_1, v_2, \ldots, v_n, o_1, o_2, \ldots, o_n)$ to (9.7). In the case where the system of linear equations has no solutions, we try new values for the vinegar variables until we find a system that does have solutions.

In order to disguise the special nature of the polynomial (9.6), we "mix" the oil and vinegar variables with the use of an affine transformation $S : (\mathbb{F}_q)^{2n} \rightarrow (\mathbb{F}_q)^{2n}$ defined by

$$S(x_1, x_2, \ldots, x_{2n}) = (x_1, x_2, \ldots, x_{2n})M + (r_1, r_2, \ldots, r_{2n}), \tag{9.8}$$

where M is a $2n \times 2n$ invertible matrix over $\mathbb{F}_q$ and $(r_1, r_2, \ldots, r_{2n}) \in (\mathbb{F}_q)^{2n}$ is a random vector. This will allow us to define public verification polynomials that appear to be more complicated than the private signing polynomials that are used to compute the signature in the first place. Note that the inverse transformation to (9.8) is simply

$$S^{-1}(y_1, y_2, \ldots, y_{2n}) = ((y_1, y_2, \ldots, y_{2n}) - (r_1, r_2, \ldots, r_{2n}))M^{-1}. \tag{9.9}$$

The *Oil and Vinegar Signature Scheme* is outlined as Cryptosystem 9.5. A signature on a message $(m_1, m_2, \ldots, m_n) \in (\mathbb{F}_q)^n$ is a vector $(s_1, s_2, \ldots, s_{2n}) \in (\mathbb{F}_q)^{2n}$ such that

$$f_k^{\mathsf{pub}}(s_1, s_2, \ldots, s_{2n}) = m_k \tag{9.10}$$

where

$$f_k^{\mathsf{pub}}(x_1, x_2, \ldots, x_{2n}) = f_k(S(x_1, x_2, \ldots, x_{2n})) \tag{9.11}$$

for all k with $1 \leq k \leq n$.[3] Verifying a signature $(s_1, s_2, \ldots, s_{2n})$ on a message $(m_1, m_2, \ldots, m_n)$ simply requires evaluating the public polynomials at the signature value to determine whether (9.10) holds as required. However, forging a signature on a message $(m_1, m_2, \ldots, m_n)$ without knowledge of the public key requires solving the system (9.10) of n multivariate quadratic equations in $2n$ variables. The security of this scheme relies on the hope that the affine transformation S can disguise the structure of the signing equations. The system (9.10) should look like a general system of multivariate quadratic equations that is presumably difficult to solve.

Signing a message $(m_1, m_2, \ldots, m_n)$ can be carried out efficiently as follows:

1. Find a solution $(v_1, v_2, \ldots, v_n, o_1, o_2, \ldots, o_n)$ to the system of equations (9.7).

2. Apply the inverse of the transformation S to this solution (as specified in (9.9)), giving $(s_1, s_2, \ldots, s_{2n}) = S^{-1}(v_1, v_2, \ldots, v_n, o_1, o_2, \ldots, o_n)$.

Note that the chosen structure of the signing polynomials means that both of these steps can be carried out efficiently using just linear algebra. This makes signing fast, which is a nice feature of this scheme.

[3] As usual, we would probably sign a message digest, rather than a message. But this is not important for the discussion of this scheme, as well as other schemes described in the following sections.

Cryptosystem 9.5: *Oil and Vinegar Signature Scheme*

Let $\mathbb{F}_q$ be a finite field and let $n > 0$ be an integer. The signing key consists of an invertible affine transformation $S : (\mathbb{F}_q)^{2n} \rightarrow (\mathbb{F}_q)^{2n}$, together with a system of n quadratic polynomials in $2n$ variables over $\mathbb{F}_q$, each of the form

$$f_k(x_1, x_2, \dots, x_{2n}) = \sum_{i=1}^{2n} \sum_{j=1}^{n} a_{ij}^k x_i x_j + \sum_{i=1}^{2n} b_i^k x_i + c^k,$$

for all k with $1 \leq k \leq n$, where a_{ij}^k, b_i^k, c^k are drawn randomly from $\mathbb{F}_q$ for all i with $1 \leq i \leq 2n$ and all j with $1 \leq j \leq n$.

The public verification key is the system of n quadratic functions in $2n$ variables, namely $f_k^{\mathsf{pub}}(x_1, x_2, \dots, x_{2n})$ for $1 \leq k \leq n$, which are defined by the formulas

$$f_k^{\mathsf{pub}}(x_1, x_2, \dots, x_{2n}) = f_k(S(x_1, x_2, \dots, x_{2n})),$$

for all k with $1 \leq k \leq n$.

Messages are elements of $(\mathbb{F}_q)^n$. A message $(m_1, \dots, m_n)$ is signed by first finding a solution $(v_1, v_2, \dots, v_n, o_1, o_2, \dots, o_n)$ to the system of equations $f_k(x_1, x_2, \dots, x_{2n}) = m_k$ for all k with $1 \leq k \leq n$. The inverse transformation S^{-1} is then applied to obtain the signature

$$(s_1, s_2, \dots, s_{2n}) = S^{-1}(v_1, v_2, \dots, v_n, o_1, o_2, \dots, o_n).$$

Verification of a signature $(s_1, s_2, \dots, s_{2n})$ on a message $(m_1, m_2, \dots, m_n)$ consists of checking that the relationship $f_k^{\mathsf{pub}}(s_1, s_2, \dots, s_{2n}) = m_k$ holds for all k with $1 \leq k \leq n$.

We can check that the resulting signature is valid by observing that

$$\begin{aligned} f_k^{\mathsf{pub}}(s_1, s_2, \dots, s_{2n}) &= f_k^{\mathsf{pub}}(S^{-1}(v_1, v_2, \dots, v_n, o_1, o_2, \dots, o_n)) \\ &= f_k(S(S^{-1}(v_1, v_2, \dots, v_n, o_1, o_2, \dots, o_n))) \\ &= f_k(v_1, v_2, \dots, v_n, o_1, o_2, \dots, o_n), \\ &= m_k, \end{aligned}$$

as required, for all k with $1 \leq k \leq n$.

Example 9.8 Let f_1 and f_2 be polynomials in four variables over $\mathbb{F}_2$ given by

$$\begin{aligned} f_1(x_1, x_2, x_3, x_4) &= x_1 x_2 + x_2 x_3 + x_4, \\ f_2(x_1, x_2, x_3, x_4) &= x_1 x_3 + x_2 x_4 + x_2. \end{aligned}$$

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

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

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 $f_1^{\mathsf{pub}}(0, 1, 1, 0)$ and $f_2^{\mathsf{pub}}(0, 1, 1, 0)$, which gives the results 0 and 1 respectively. $\square$

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 $f : Y \rightarrow Z$ 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 $f : \{0, \ldots, p-2\} \rightarrow \mathbb{Z}_p{}^*$. 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. $\square$

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 $(x_1, \dots, x_k)$ is a message and $(y_{1,x_1}, \dots, y_{k,x_k})$ is its signature. Now suppose Oscar tries to sign the new message $(x'_1, \dots, x'_k)$. 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 $(x'_1, \dots, x'_k)$.

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***

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

according to the rules $y^j = f(y^{j-1})$ for $1 \leq j \leq 7$, and $z = f(y_7)$. We can equivalently define $y^j = f^j(y_0)$ for $1 \leq j \leq 7$, 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, $y^0, \dots, y^{2^w-1}, z$.

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 $y_1^0, y_2^0, \dots, y_\ell^0$. These initial values comprise the private key.

Now consider a message $(x_1, \dots, x_\ell)$, 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 $i = 1, \dots, \ell$ 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:

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

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

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

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 $a_3 = y_3^1$:

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

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

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

The verification requires checking that

$$\begin{aligned} f^5(a_1) &= z_1, \\ f^3(a_2) &= z_2, \quad \text{and} \\ f^7(a_3) &= z_3. \end{aligned}$$

□

The above process is quite ingenious, but it is not secure. Let us look at Example 9.10 to see what the problem is. Consider the signature (a_1, a_2, a_3) given in this example. The released values are just elements in the three hash chains, and once an element in a hash chain is known, anyone can compute any later values in the hash chains, as desired. So, for example, Oscar could compute

$$\begin{aligned} y_1^5 &= f^2(a_1), \\ y_2^6 &= f(a_2), \quad \text{and} \\ y_3^4 &= f^3(a_3). \end{aligned}$$

Therefore, Oscar can now create the signature (y_1^5, y_2^6, y_3^4) for the message 101110100.

Fortunately, a small tweak will yield a secure signature scheme. The fix is to include a ***checksum*** in the message, and also sign the checksum. The checksum is defined to be

$$C = \sum_{i=1}^{\ell} (2^w - 1 - x_i).$$

In Example 9.10, we would have

$$C = (7-3) + (7-5) + (7-1) = 4 + 2 + 6 = 12.$$

In binary, we have $C = 1100$. After padding on the left with two zeroes, we can break C into two chunks of three bits: $x_4 = 001$ and $x_5 = 100$. Now we create two additional hash chains and use them to sign x_4 and x_5, releasing the values $a_4 = y_4^1 = f(y_4)$ and $a_5 = y_5^4 = f^4(y_5)$. These two hash chains have public keys z_4 and z_5, respectively.

Pictorially, we have

$$y_4^0 \to \boxed{y_4^1} \to y_4^2 \to y_4^3 \to y_4^4 \to y_4^5 \to y_4^6 \to y_4^7 \to z_4$$

$$y_5^0 \to y_5^1 \to y_5^2 \to y_5^3 \to \boxed{y_5^4} \to y_5^5 \to y_5^6 \to y_5^7 \to z_5.$$

So the entire signature on the message (x_1, x_2, x_3) is $(a_1, a_2, a_3, a_4, a_5)$. To verify this signature, the following steps are performed:

1. Verify that (a_1, a_2, a_3) is the correct signature for (x_1, x_2, x_3).
2. Form the checksum and create (x_4, x_5).
3. Verify that (a_4, a_5) is the correct signature for (x_4, x_5).

We now argue informally that the signature scheme is secure when we include a checksum as described above. Suppose that Oscar sees a message (x_1, x_2, x_3) and its signature $(a_1, a_2, a_3, a_4, a_5)$ (where a_4 and a_5 comprise the signature on the checksum). Oscar then wants to create a signature on a second message (x'_1, x'_2, x'_3). Since Oscar can only move "forward" in the hash chains, it must be the case that $x'_i \geq x_i$ for $i = 1, 2, 3$. Also, because $(x'_1, x'_2, x'_3) \neq (x_1, x_2, x_3)$, it follows that $x'_{i_0} > x_{i_0}$ for some i_0. From this, we see that $C' < C$, where C' is the checksum for (x'_1, x'_2, x'_3). This means that $x'_4 < x_4$ or $x'_5 < x_5$ (or both). For purposes of illustration, suppose that $x'_4 < x_4$. Then Oscar cannot sign x'_4 because he would need to move "backwards" in the hash chain, which is not possible due to the one-way property of f. A similar contradiction arises if $x'_5 < x_5$.

In general, the checksum C will satisfy the inequality $0 \leq C \leq \ell(2^w - 1)$. The number of bits in the binary representation of C is at most $w + \log_2 \ell$. Let B denote the number of w-bit blocks that are required to store C. Then

$$B \leq 1 + \left\lceil \frac{\log_2 \ell}{w} \right\rceil .$$

So we will have ℓ message blocks and B checksum blocks, giving rise to a total of $\ell + B$ hash chains.

Note that the value of w should be chosen carefully. As w increases, the number of hash chains decreases. However, the time to create and verify signatures increases, because we have to traverse longer hash chains.

There is one other improvement that can be made to shrink the size of the public key. Instead of using the tuple $(z_1, z_2, z_3, z_4, z_5)$ as the public key, we concatenate these values and pass them through as secure cryptographic hash function, say h. Thus, the public key is defined to be $\mathbf{z} = h(z_1 \parallel z_2 \parallel z_3 \parallel z_4 \parallel z_5)$. The verification of the signature would comprise the following steps:

1. compute the ends of all the hash chains,
2. concatenate the results,
3. apply the hash function h,
4. compare the output of h to the public key $\mathbf{z}$.

We summarize by giving a description of the *Winternitz Signature Scheme* for arbitrary values of w in Cryptosystem 9.7.

Cryptosystem 9.7: *Winternitz Signature Scheme*

Let k and w be positive integers, where $\ell = k/w$ is an integer, and define

$$B = 1 + \left\lceil \frac{\log_2 \ell}{w} \right\rceil .$$

Suppose $f : Y \rightarrow Y$ is a one-way function and suppose $h : Y \rightarrow Z$ is a secure hash function.

Construct $k + B$ hash chains using f, each of length 2^w. The starting points of the hash chains (which comprise the private key) are y_i^0 and the ending points are z_i, for $1 \leq i \leq k + B$. The public key is

$$\mathbf{z} = h(z_1 \parallel z_2 \parallel \cdots \parallel z_{\ell+B}).$$

Let $\mathcal{P} = (\{0,1\}^w)^\ell$ and let $\mathcal{A} = Z$.

For a message $(x_1, \ldots, x_k) \in \mathcal{P}$, the signature $\mathbf{sig}_K(x_1, \ldots, x_k)$ is computed as follows:

1. Compute the checksum $C = (x_{k+1}, \ldots, x_{k+B})$.
2. For $1 \leq i \leq \ell + B$, compute $a_i = f^{x_i}(y_i)$.
3. The signature $\mathbf{sig}_K(x_1, \ldots, x_k) = (a_1, \ldots, a_{\ell+B})$.

A signature $(a_1, \ldots, a_{\ell+B})$ on the message $(x_1, \ldots, x_\ell)$ is verified as follows:

1. Compute the checksum $C = (x_{k+1}, \ldots, x_{k+B})$.
2. For $1 \leq i \leq \ell + B$, compute $z_i = f^{2^w - x_i}(a_i)$.
3. Check to see if $\mathbf{z} = h(z_1 \parallel z_2 \parallel \cdots \parallel z_{\ell+B})$.

9.5.3 Merkle Signature Scheme

The signature schemes in Sections 9.5.1 and 9.5.2 are one-time schemes. Merkle invented a useful method of extending a one-time scheme so it could be used for a large (but fixed) number of signatures, without increasing the size of the public key. We describe Merkle's technique in this section.

The basic idea is to create a binary tree (which is now called a ***Merkle tree***) by hashing combinations of various public keys (i.e., verification keys) of one-time signature schemes.[4] The particular one-time scheme that is used is not important.

[4]The Merkle tree will be used only to authenticate public keys; it is not used to create signatures in the component one-time signature schemes.