Chapter 15

Post-Quantum Cryptography

VERSION = 21-OCT-2025

Contents

Many widely-used cryptographic schemes—including RSA and ECC—are known to be vulnerable to attacks by quantum computers. This chapter considers two standardized families of public-key cryptosystems that are currently being deployed to mitigate the quantum threat: hash-based signatures and lattice-based schemes for key encapsulation and signatures.

15.1 The Threat of Quantum Computers

15.1.1 Quantum computing

The idea of harnessing the counterintuitive properties of quantum mechanics for computation was independently formulated by mathematician Yuri Manin and physicist Richard Feynman in the 1960s. These properties include *superposition*, *interference*, and *entanglement*. While a full discussion of quantum computing is beyond the scope of this book, we will provide a high-level overview to illustrate both the potential power and the limitations of quantum computers.

The quantum analogue of a classical bit is a *qubit*. Unlike a classical bit, which exists in one of two states (0 or 1), a qubit can exist in both states at the same time, each with a specific probability. This phenomenon is known as superposition. More generally, an n-qubit register can be in all 2^n possible binary states at once, each with a certain

probability. The potential power of quantum computers arises from the fact that when a function $f : \{0,1\}^n \longrightarrow \{0,1\}^n$ is applied to an n-qubit register, it is evaluated on all 2^n states simultaneously in a single computational step. In contrast, a classical computer, would need to evaluate the function on each input state sequentially, requiring 2^n separate function evaluations.

This remarkable capability makes quantum computers at first glance appear to function as massively parallel machines. However, this interpretation is misleading since one is unable to view the result of applying f to all the 2^n states. Instead, when an n-qubit register is "measured", it collapses into one of the 2^n possible states according to the underlying probability distribution. Thus, the only observable output is $f(x)$ for a single input x selected according to the aforementioned probability distribution.

The challenge in designing quantum algorithms is to harness properties of superposition, interference, and entanglement so that, at the end of a computation, the desired solution to a problem is obtained with probability very close to 1. Thus, when the qubit register is measured at the end of the computation, the desired solution is observed with high confidence.

The potential power of quantum computers became evident in the 1990s with the discovery of Shor's algorithm, which efficiently factors large numbers, and Grover's algorithm, which provides a quadratic speedup for unstructured search problems.

15.1.2 Shor's algorithm

As of 2025, three families of public-key cryptographic schemes are widely used in practice: (i) RSA, whose security relies on the difficulty of factoring integers n; (ii) discrete logarithm systems which rely on the hardness of the discrete logarithm problem in the multiplicative group of integers modulo a prime p; and (iii) ECC, which depends on the hardness of the elliptic curve discrete logarithm problem (ECDLP) in an elliptic curve group $E(\mathbb{Z}_p)$. The fastest classical algorithms known for factoring integers and computing discrete logarithms modulo a prime have subexponential time complexity, while solving ECDLP requires fully exponential running time.

In 1994, Peter Shor discovered a quantum algorithm capable of solving all three problems in polynomial time. Specifically, Shor's algorithm requires $O(\log n)^2$, $O(\log p)^2$, and $O(\log p)^2$ quantum operations, for factoring, discrete logarithms, and elliptic curve discrete logarithms, respectively. As a result, all RSA, discrete logarithm, and elliptic curve cryptographic schemes would be completely broken if and when large-scale quantum computers are built.

15.1.3 Grover's algorithm

The *unstructured search problem* is defined as follows. Let $F : \{0,1\}^n \longrightarrow \{0,1\}$ be a function that (i) is efficiently computable; and (ii) satisfies $F(x) = 1$ for exactly d inputs $x \in \{0,1\}^n$, where $1 \leq d \ll 2^n$. The goal is to find an $x \in \{0,1\}^n$ such that $F(x) = 1$.

The fastest possible classical algorithm for the unstructured search problem is a random sampling algorithm, which repeatedly selects x uniformly at random from $\{0,1\}^n$ and evaluates $F(x)$ until $F(x) = 1$ is observed. The expected number of function evaluations is $2^n/d$.

In 1996, Lov Grover discovered a quantum algorithm that solves the unstructured search problem using only $\sqrt{2^n/d}$ function evaluations. It's important to note that Grover's algorithm is not amenable to linear speedup parallelization—if m quantum computers are used simultaneously, then the time to find a solution is reduced by a factor of only $\sqrt{m}$ rather than m.

Quantum attack on block ciphers. Let E be the encryption algorithm of a block cipher with ℓ-bit key. Suppose we are given t plaintext-ciphertext pairs (m_i, c_i), where each ciphertext is computed as $c_i = E_{k'}(m_i)$ for some secret key k'. If t is sufficiently large, then with probability very close to 1, there is only one k satisfying $c_i = E_k(m_i)$ for all $1 \leq i \leq t$. Brute-force key search recovers the secret key k' using $2^{\ell-1}$ encryption operations on average, and 2^ℓ in the worst case. However, Grover's algorithm offers a faster quantum alternative.

Define the function $F : \{0,1\}^\ell \longrightarrow \{0,1\}$ by $F(k) = 1$ if $E_k(m_i) = c_i$ for all $1 \leq i \leq t$, and $F(k) = 0$ otherwise. Grover's algorithm (with $d = 1$) finds k' such that $F(k') = 1$ using $2^{\ell/2}$ quantum evaluations of E, which is significantly faster than the classical brute-force approach.

While Grover's algorithm is faster, determining whether it's more cost-effective than classical brute-force for recovering the secret key of a block cipher is challenging. Nonetheless, the threat posed by Grover's algorithm to the security of the block cipher AES can be mitigated by employing 256-bit keys instead of 128-bit keys; this increase in key size comes with a modest increase in encryption and decryption time.

15.1.4 Quantum-safe cryptography

Factoring a 2048-bit RSA number using Shor's algorithm would require a quantum computer with at least 2048 qubits, while solving a 256-bit ECDLP instance would require at least 256 qubits. Most importantly, these qubits must be *fault tolerant*; this requirement is due to "quantum decoherence", a phenomenon in which qubits interact with their environment, introducing errors that cause them to lose their quantum properties.

Several proof-of-concept quantum computers have been built, including a 1,121-qubit machine by IBM in 2023. However, the existing qubits are *not* fault tolerant and are not capable of solving cryptographically interesting instances of integer factorization and ECDLP. In fact, as of 2025, the largest number factored using Shor's algorithm on a quantum computer is the number 21.

Researchers have been designing quantum error correcting codes, which combine multiple (imperfect) *physical* qubits to form a single *logical* (i.e., fault-tolerant) qubit. By continuously measuring and correcting a fraction of the physical qubits every few nanoseconds, they aim to maintain a stable logical qubit that is almost error-free.

Estimates suggest that several thousand physical qubits may be needed to build a

single reliable logical qubit. As a result, factoring a 2048-bit RSA modulus or solving a 256-bit ECDLP instance using Shor's algorithm may require millions of physical qubits. While recent breakthroughs in quantum error correction are promising, it remains uncertain if, and when, large-scale fault-tolerant quantum computers capable of breaking RSA or ECC will be built. Thus, it remains unclear how imminent the threat posed by quantum computers truly is.

On the other hand, there is no fundamental reason (yet) known why a cryptographically-relevant quantum computer *cannot* be built. If such machines are built, then most internet security protocols that rely on public-key cryptography would be compromised, including automatic software updates (which rely on digital signatures) and TLS (which relies on digital signatures and elliptic curve Diffie-Hellman). Moreover, organizations such as the U.S. National Security Agency (NSA), are already collecting and storing vast amounts of encrypted internet traffic for so-called Harvest-Now, Decrypt Later (HNDL) attacks. Even if this data had been securely encrypted, NSA will have the ability to decrypt the data if and when they can build cryptographically-relevant quantum computers.

It's difficult to decide what one should do, if anything at all, to mitigate the quantum threat, and also when to take action. Nevertheless, interest in deploying quantum-resistant public-key cryptography grew dramatically in the latter 2010s, with a surge in August 2024 when the U.S. National Institute of Standards and Technology (NIST) published standards for quantum-safe signatures and key encapsulation. These standards specify the lattice-based key encapsulation scheme Kyber-KEM (ML-KEM), the lattice-based signature scheme Dilithium (ML-DSA), and the stateless hash-based signature scheme SPHINCS+ (SLH-DSA). Two stateful hash-based signature schemes, the Leighton-Micali Scheme (LMS) and the eXtended Merkle Signature Scheme (XMSS) had already been standardized by NIST in 2020.

Governments worldwide are now promoting standards for quantum-safe cryptography, and industry is beginning to integrate quantum-safe cryptography into their products. We'll consider LMS and SPHINCS+ in §15.2, Kyber-KEM in §15.3, and Dilithium in §15.4.

15.2 Hash-Based Signature Schemes

Hash-based signatures schemes are appealing due to their straightforward security analysis against both classical and quantum attacks. Their security relies only on well-established properties of hash functions, and not on newer or less-tested number-theoretic assumptions. In §15.2.1, we introduce a conceptually simple hash-based signature scheme proposed by Lamport in 1975. Lamport's signature scheme is called a one-time signature scheme (OTS) because it has the undesirable property that a key pair can securely sign only one message. We'll examine several drawbacks with Lamport's scheme in §15.2.2 and discuss solutions. These solutions were incorporated into the stateful Leighton-Micali signature scheme (LMS), which we'll describe in §15.2.3. LMS has been specified

by the Internet Engineering Task Force (IETF) and later standaridized by the National Institute of Standards and Technology (NIST). Lastly, §15.2.4 outlines the main ideas behind SPHINCS+, a stateless hash-based signature scheme also standardized by NIST.

15.2.1 Lamport one-time signature scheme

We'll begin by describing a signature scheme designed for signing one-bit messages. Let $G : \{0,1\}^n \rightarrow \{0,1\}^n$ be a preimage-resistant hash function that accepts fixed-size messages of n bits. Alice randomly selects two bitstrings $x_0, x_1 \in \{0,1\}^n$ and computes their hash values $y_0 = G(x_0)$ and $y_1 = G(x_1)$. Her (signing) private key is $X = (x_0, x_1)$ and her (verification) public key is $Y = (y_0, y_1)$. To sign a message $m \in \{0,1\}$, Alice sets her signature as $S = x_m$, i.e., $S = x_0$ if $m = 0$, and $S = x_1$ if $m = 1$. Anyone who possesses an authentic copy of Alice's public key Y can verify the signed message (m, S) by checking whether $y_m = G(S)$. The security of this scheme relies on the difficulty of finding preimages for G, i.e., computing an $x'_i \in \{0,1\}^n$ such that $y_i = G(x'_i)$.

Lamport's one-time signature scheme (Alg. 15.1) extends this one-bit signing mechanism to messages of arbitrary bitlengths by signing the individual bits of a message's hash value. Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be a collision-resistant hash function. For each

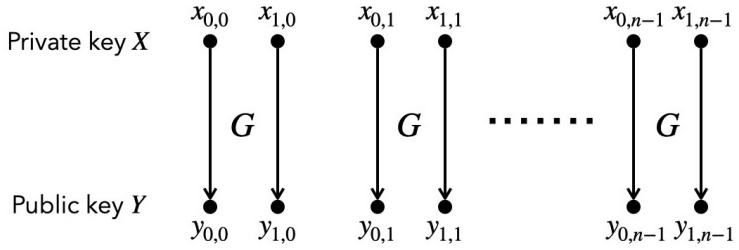

Figure 15.1: A private key X and public key Y for Lamport's OTS.

$i \in \{0,1\}$ and $0 \leq j \leq n-1$, Alice selects $x_{i,j} \in_R \{0,1\}^n$ for a private key $X = (x_{0,0}, x_{1,0}, x_{0,1}, x_{1,1}, \ldots, x_{0,n-1}, x_{1,n-1})$. Thus X consists of $2n$ random n-bit strings. Using G as above, her public key Y consists of the hash values $y_{i,j} = G(x_{i,j})$. Thus, $Y = (y_{0,0}, y_{1,0}, y_{0,1}, y_{1,1}, \ldots, y_{0,n-1}, y_{1,n-1})$; see Figure 15.1. To sign a message $M \in \{0,1\}^*$, Alice first computes its hash $h = H(M)$. The signature on M is $S = (s_0, s_1, \ldots, s_{n-1})$, where $s_j = x_{h_j,j}$ and h_j is the jth bit of h. The signed message (M, S) can be verified by first computing $h = H(M)$, and then checking that the signature components are correct, i.e., $G(s_j) = y_{h_j,j}$ for $0 \leq j \leq n-1$.

Observe that when Alice signs a message M, the signature S exposes half the components of her private key X, specifically the $x_{i,j}$ values for which $h_j = i$. Suppose now that Alice signs a second message M'. One expects that about half the bits in $H(M')$ will differ from those in $H(M)$, and consequently Alice's signature S' on M' is likely to reveal an additional one-quarter of the private key components. Thus, to maintain security, Alice should sign only one message using the same key. This is why Lamport's signature

Algorithm 15.1: Lamport's one-time signature scheme (Lamport's OTS)

Domain parameters: Hash functions $G : \{0,1\}^n \rightarrow \{0,1\}^n$ and $H : \{0,1\}^* \rightarrow \{0,1\}^n$.

Key generation. Alice generates a private key X and public key Y as follows.

1 Select $x_{i,j} \in_R \{0,1\}^n$ for $i = 0, 1$ and $0 \leq j \leq n-1$.
2 Set $X = (x_{0,0}, x_{1,0}, x_{0,1}, x_{1,1}, \ldots, x_{0,n-1}, x_{1,n-1})$.
3 Compute $y_{i,j} = G(x_{i,j})$ for $i = 0, 1$ and $0 \leq j \leq n-1$.
4 Set $Y = (y_{0,0}, y_{1,0}, y_{0,1}, y_{1,1}, \ldots, y_{0,n-1}, y_{1,n-1})$.
5 **return** (X, Y).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following.

6 Compute $h = H(M)$ and write $h = (h_0, h_1, \ldots, h_{n-1})$ where each $h_j \in \{0,1\}$.
7 Let $S = (s_0, s_1, \ldots, s_{n-1})$ where $s_j = x_{h_j,j}$.
8 **return** (S).

Signature verification. To verify Alice's signature S on a message M, Bob does:

9 Obtain an authentic copy of Alice's public key Y.
10 Compute $h = H(M)$ and write $h = (h_0, h_1, \ldots, h_{n-1})$ where each $h_j \in \{0,1\}$.
11 **for** $j = 0$ **to** $n-1$ **do**
12 If $G(s_j) \neq y_{h_j,j}$ then **return** (REJECT)
13 **return** (ACCEPT).

scheme is a *one-time signature scheme*.

The security notion for one-time signature schemes is a specialization of the security notion for general signature schemes (Definition 7.3). In this model, an adversary is given Alice's public key Y, and permitted to request from Alice her signature S on one message M of the adversary's choosing. The scheme is said to be *secure* if such an adversary is unable to produce a single forgery, i.e., a signed message (M^*, S^*), with $M^* \neq M$, that would be accepted by Alice's verification algorithm.

It is relatively straightforward to show that Lamport's one-time signature scheme remains secure as long as the hash function H is collision resistant and G is preimage resistant (Exercise 15.3). The scheme has a security level of $n/2$ bits against quantum attacks since the fastest method known for finding preimages for G is Grover's quantum algorithm (Exercise 15.1) which runs in time $O(2^{n/2})$, and the most cost-effective method—whether classical or quantum—for finding collisions for H is the van Oorschot-Wiener parallel collision search method, which also runs in time $O(2^{n/2})$.

To attain a 128-bit security level against both classical and quantum attacks, the Lamport signature scheme can be implemented with $n = 256$. In this setup, public keys and private keys each have size 16,384 bytes, whereas a signature has size 8,192 bytes.

15.2.2 Limitations of Lamport's signature scheme, and remedies

In this section, we'll highlight four issues with Lamport's signature scheme, and introduce solutions that lead to the Leighton-Micali signature scheme in §15.2.3.

Issue #1: Large private keys. A private key requires $2n^2$ bits of storage, or 16,384 bytes when $n = 256$. To reduce this, the private key components $x_{i,j}$ can be generated from a randomly selected n-bit secret seed and a counter using a pseudorandom function. The seed is securely stored and used to generate the private key components when needed. This reduces the storage for the private key to n bits, or 32 bytes when $n = 256$.

Issue #2: Large public keys and signatures. A public key requires $2n^2$ bits of storage, while a signature has bitlength n^2. These large sizes are a consequence of the bits of the hash value $H(M)$ being signed one at a time. In 1979, Winternitz proposed using hash chains to sign the bits of $H(M)$ one w-bit block at a time. This results in smaller public keys and signatures, at the expense of slower signature generation and verification.

An integer divisor w of n is selected; w is called the *Winternitz parameter*. Let

$$\ell_1 = n/w, \quad \ell_2 = \lfloor \log_2(\ell_1(2^w - 1))/w \rfloor + 1, \quad \text{and} \quad \ell = \ell_1 + \ell_2. \tag{15.1}$$

ℓ_1 is the number of w-bit blocks in $H(M)$, called the block length of $H(M)$, and ℓ_2 is the block length of the checksum defined in (15.2). Define the i-fold iteration of G by $G^0(x) = x$ and $G^i(x) = G(G^{i-1}(x))$ for $1 \le i \le 2^w - 1$. The sequence $G^0(x), G^1(x), \ldots, G^{2^w-1}(x)$ is a *hash chain* of length $2^w - 1$ starting at x and ending at x_{2^w-1}; see Figure 15.2. Given any element x_i on a hash chain, it's easy to compute all elements x_j with $j > i$ by repeatedly applying G to x_i. However, computing x_j for any $j < i$ is infeasible assuming that G is preimage resistant.

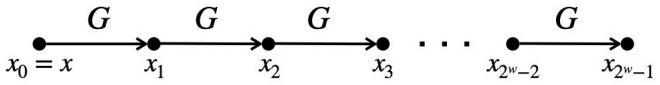

Figure 15.2: A Winternitz hash chain of length $2^w - 1$ starting at x.

In Winternitz's one-time signature scheme, Alice's private key consists of ℓ randomly-chosen bitstrings $x_0, x_1, \ldots, x_{\ell-1}$ each of length n. Her corresponding public key consists of the bitstrings $y_i = G^{2^w-1}(x_i)$ for $0 \le i \le \ell - 1$. To sign a message $M \in \{0,1\}^*$, Alice computes $h = H(M)$ and splits the n bits of h into w-bit blocks h_i, forming $h = (h_0, h_1, \ldots, h_{\ell_1-1})$. She also computes the following *checksum* u of h:

$$u = \text{csum}(h) = \sum_{i=0}^{\ell_1-1} (2^w - 1 - h_i), \tag{15.2}$$

where each w-bit block h_i is interpreted as a non-negative integer in $[0, 2^w - 1]$. Now, the base-2^w representation of u has $\lfloor \log_{2^w} u \rfloor + 1 = \lfloor (\log_2 u)/w \rfloor + 1$ digits, since $\log_{2^w}(x) =$

$\log_2(x)/w$. Observe that $0 \le u \le \ell_1(2^w - 1)$ since $0 \le h_i \le 2^w - 1$. Thus, the number of base-2^w digits in u is at most ℓ_2.

It follows that u can be represented as a binary number $\tilde{c}(h)$ of bitlength exactly $\ell_2 w$, by adding leading 0's if necessary. The bits of $\tilde{c}(h)$ are then divided into ℓ_2 w-bit blocks $h_{\ell_1}, h_{\ell_1+1}, \ldots, h_{\ell_1+\ell_2-1}$. Alice's signature on M is $S = (s_0, s_1, \ldots, s_{\ell-1})$, where $s_i = G^{h_i}(x_i)$ for $0 \le i \le \ell - 1$. Thus, the block value h_i defines the number of iterations of G for the ith signature block.

To verify Alice's signature S on M, anyone with an authentic copy of Alice's public key can compute $h = H(M)$, extract the w-bit blocks $h_0, h_1, \ldots, h_{\ell-1}$ from h and u as was done in signature generation, and then confirm that $y_i = G^{2^w-1-h_i}(s_i)$ for $0 \le i \le \ell - 1$. A properly generated signed message (M, S) will be accepted by a verifier since

$$G^{2^w-1-h_i}(s_i) = G^{2^w-1-h_i}(G^{h_i}(x_i)) = G^{2^w-1}(x_i) = y_i \text{ for each } 0 \le i \le \ell - 1.$$

Example 15.1 (*Winternitz with toy parameters*) The Winternitz signature scheme is illustrated here with toy parameters $n = 12$, $w = 3$, $\ell_1 = 4$, $\ell_2 = 2$ and $\ell = 6$. Figure 15.3(a) depicts a public-private key pair. Suppose Alice wishes to sign a message M whose hash

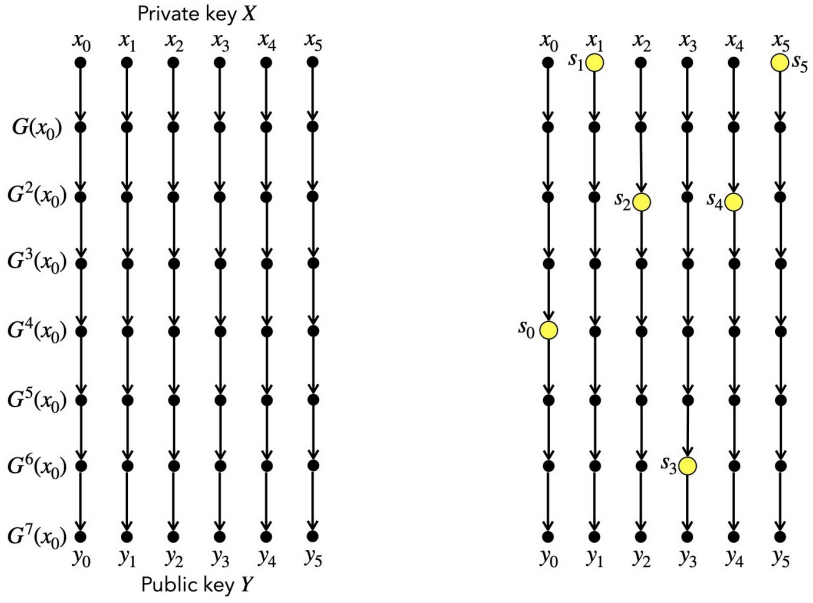

Figure 15.3: (a) A Winternitz public-private key pair (left); (b) The Winternitz signature $S = (s_0, s_1, s_2, s_3, s_4, s_5)$ on a message M with $H(M) = 100000010110$ (right).

(as a binary string) is $h = H(M) = 100000010110 \in \{0,1\}^{12}$. The hash value h is split into 3-bit blocks as (100, 000, 010, 110), and the checksum $u = \text{csum}(h)$ is computed as $u = \text{csum}(h) = (7-4) + (7-0) + (7-2) + (7-6) = 16$. This checksum is encoded in binary as $\tilde{c}(h) = 010000$, which is further divided into two additional 3-bit blocks (010, 000). Alice's signature on M is $S = (G^4(x_0), G^0(x_1), G^2(x_2), G^6(x_3), G^2(x_4), G^0(x_5))$,

as depicted in Figure 15.3(b). To verify the signature, the recipient computes $h = H(M)$ and checks that $y_0 = G^3(s_0)$, $y_1 = G^7(s_1)$, $y_2 = G^5(s_2)$, $y_3 = G(s_3)$, $y_4 = G^5(s_4)$, and $y_5 = G^7(s_5)$.

The crux of the security argument for the Winternitz OTS is that if one is given a value $s_i = G^{h_i}(x_i)$ in the ith hash chain, then one can certainly compute $G^{h_i^*}(x_i)$ for all $h_i^* > h_i$ (by repeatedly applying G to s_i). However, one cannot compute (as an attacker may wish to, in order to forge a signature) $G^{h_i^*}(x_i)$ for any $h_i^* < h_i$ since G is preimage resistant.

Here is an informal security argument. Suppose that a forger has a single valid signed message (M, S), where $S = (s_0, s_1, \ldots, s_{\ell-1})$. Suppose also that the forger is able to produce a valid signed message (M^*, S^*) where $S^* = (s_0^*, s_1^*, \ldots, s_{\ell-1}^*)$ and $M^* \neq M$. Since H is collision resistant, we can assume that $H(M^*) \neq H(M)$—otherwise the forger would have found a collision for H. Let $H(M) = h = (h_0, \ldots, h_{\ell_1-1})$ and $H(M^*) = h^* = (h_0^*, \ldots, h_{\ell_1-1}^*)$. Because $h^* \neq h$, there must be at least one index $i \in [0, \ell_1 - 1]$ for which $h_i^* \neq h_i$. If $h_i^* < h_i$, then the forger was somehow able to compute $s_i^* = G^{h_i^*}(x_i)$ from $s_i = G^{h_i}(x_i)$, which we noted is not possible since G is preimage resistant. Thus, $h_i^* > h_i$ and $h_j^* \geq h_j$ for all $j \in [1, \ell_1]$ with $j \neq i$, from which we conclude that $\text{csum}(h^*) < \text{csum}(h)$. Let $\tilde{c}(h) = (h_{\ell_1}, \ldots, h_{\ell-1})$ and $\tilde{c}(h^*) = (h_{\ell_1}^*, \ldots, h_{\ell-1}^*)$. Since $\text{csum}(h^*) < \text{csum}(h)$, there must be at least one index $k \in [\ell_1, \ell - 1]$ for which $h_k^* < h_k$. But again, this means that the forger was able to compute $s_k^* = G^{h_k^*}(x_k)$ from $s_k = G^{h_k}(x_k)$ where $h_k^* < h_k$, which is not possible since G is preimage resistant. Thus, the forger cannot produce the valid signature for a message different from M.

Winternitz public keys and signatures have a size of ℓn bits. For instance, when $n = 256$ and $w = 8$, implying $\ell_1 = 32$, $\ell_2 = 2$ and $\ell = 34$, the public key and signature sizes are 1,088 bytes. This is a substantial reduction compared to Lamport's scheme, which requires 16,384-byte public keys and 8,192-byte signatures when $n = 256$.

Issue #3: A private signing key can be used only once. Like the Lamport signature scheme, the Winternitz scheme is a one-time signature scheme since signing multiple messages exposes progressively more of the private key—in the case of Winternitz, longer segments of the hash chains. Since security deteriorates rapidly with each additional signature, the private key should be used only once. However, an OTS is impractical for most applications due of the significant overhead of securely distributing public verification keys, e.g., through certificates.

To mitigate this issue, Merkle hash trees are employed, per Example 15.2. An entity, say Alice, generates $N = 2^d$ key pairs (X_i, Y_i), $0 \leq i \leq N - 1$, for an OTS. A complete binary tree of height d is built where each leaf is assigned the value $F(Y_i)$, the hash of Y_i using a collision-resistant n-bit hash function F. Each internal node's value is derived by hashing the concatenation of the values of its two children. The root's value R serves as Alice's long-term public key, which she securely distributes, e.g., via a certificate.

Alice can sign up to N messages, using the private key X_i to generate a signature S_i for her ith message M_i. She includes Y_i with the signature, along with the values

associated with all sibling nodes along the path in the tree from the ith leaf to the root; this sequence is called an *authentication path*. A verifier can confirm the authenticity of public key Y_i by computing $F(Y_i)$ and then using the provided hash values to reconstruct a root value R'. The authenticity of Y_i is confirmed if $R' = R$, thanks to the collision resistance of F. This approach allows Alice to sign up to N messages, while securely distributing only a single public key R.

Example 15.2 (*Merkle tree of height $d = 3$*) Figure 15.4 depicts a Merkle tree of height $d = 3$. The path from the 5th leaf to the root is $h_{12} \longrightarrow h_6 \longrightarrow h_3 \longrightarrow R$. The signature

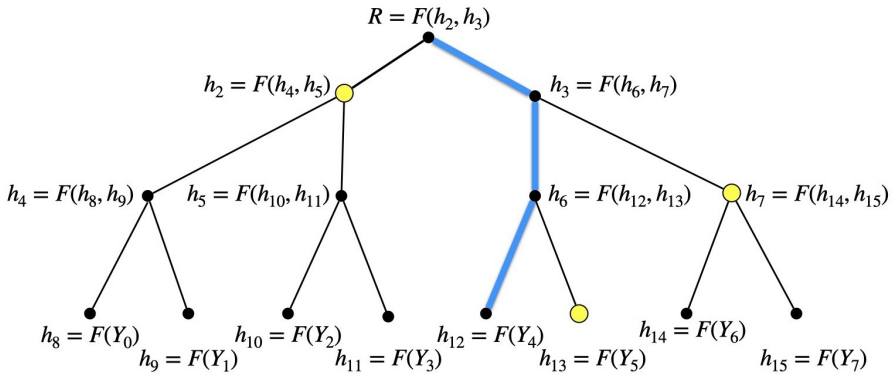

Figure 15.4: A Merkle tree of height $d = 3$, and an authentication path from h_{12} to R.

for the fifth message M_4 includes the public verification key Y_4, and the sibling hash values h_{13}, h_7 and h_2. A recipient who possesses an authentic copy of R can confirm the authenticity of Y_4 by computing $h_{12} = F(Y_4)$, $h_6 = F(h_{12}, h_{13})$, $h_3 = F(h_6, h_7)$, and verifying that $R = F(h_2, h_3)$.

Using a Merkle tree reduces the public key size significantly, but increases the signature size. For example, suppose that a Merkle tree of height $d = 20$ is used to authenticate $N = 2^{20}$ Winternitz public keys with parameters $n = 256$ and $w = 8$. The public key R is only 32 bytes. However, a signature now consists of an 8,192-byte Winternitz signature S_i, an 8,192-byte Winternitz public key Y_i, and $d = 20$ 32-byte hash values, for a total size of 17,024 bytes.

Issue #4: Constructing a large Merkle tree can be slow. The root of a height-d Merkle tree can serve as a long-term public key, allowing the owner to sign up to $N = 2^d$ messages. To support a large number of signatures, d can be chosen to be sufficiently large, e.g., $d = 40$. However, key generation becomes very slow because it requires computing N OTS key pairs and constructing the entire height-d Merkle tree in order to determine the root value R.

To speed up key generation, a *hypertree* (also called a *multi-tree*) can be employed. Figure 15.5 depicts a two-layer hypertree. Of course, a hypertree with more than two

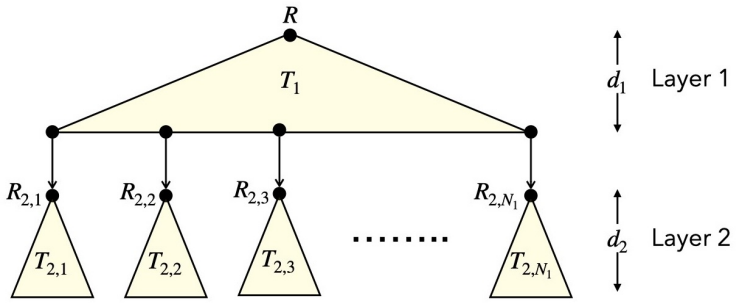

Figure 15.5: A two-layer hypertree.

layers can be used. The OTS public keys at the leaves of the layer-1 Merkle tree T_1 are used to sign the roots $R_{2,i}$ of the layer 2 Merkle trees $T_{2,i}$. The tree T_1 has height d_1, whence T_1 has $N_1 = 2^{d_1}$ leaves, whereas the layer-2 trees $T_{2,i}$ each have height d_2 and $N_2 = 2^{d_2}$ leaves. The OTS public keys at the leaves of the layer-2 trees are used to sign up to $N = 2^{d_1+d_2}$ messages. The root R of T_1 serves as the long-term public key for the signature scheme. If the OTS key associated with a leaf in tree $T_{2,i}$ is used to sign a message, then the signature should include the corresponding OTS public key, the root $R_{2,i}$ of $T_{2,i}$, the authentication path in $T_{2,i}$ to $R_{2,i}$, and the signature on $R_{2,i}$ using the ith leaf in T_1 (this signature includes the public key associated with that leaf, and the authentication path in T_1 to the root R). The verifier uses these values to confirm the authenticity of the OTS public key used to sign the message.

A key advantage of using a two-layer hypertree is that only T_1 needs to be generated in order to derive the long-term public key R, significantly reducing key generation time at the outset. The layer-2 trees can be constructed on demand as greater numbers of signatures are required. Additionally, the hypertree structure allows layer-2 trees to be generated across different cryptographic modules. Each cryptographic module can sign messages using the OTS private keys associated with leaves of its layer-2 trees. This facilitates the distribution of signing privileges across distinct cryptographic modules while minimizing the risk of OTS key reuse. A drawback of hypertrees is that signatures are larger and signature verification is slower (Exercise 15.6). As we'll see, hypertrees are used with the LMS and SPHINCS+ signature schemes.

15.2.3 Leighton-Micali signature scheme (LMS)

The Leighton-Micali signature scheme (LMS) utilizes a Merkle hash tree structure, whose leaves correspond to public keys of the Leighton-Micali one-time signature scheme (LM-OTS). LM-OTS is essentially the Winternitz scheme, with several modifications to enhance security. LM-OTS and LMS were specified by the IETF in 2019, and later standarized by NIST in 2020. These specifications also define a hypertree variant of LMS, known as the Hierarchical Signature Scheme (HSS).

LM-OTS. LM-OTS is described in Algorithm 15.2. It uses an n-bit hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$. Its parameters include the Winternitz parameter w, the number ℓ_1 of w-bit blocks in the hashed message h, the number ℓ_2 of w-bit blocks in the checksum $\tilde{c}(h)$, and the number $\ell = \ell_1 + \ell_2$ of Winternitz chains in a key pair.

To eliminate the need for collision resistance[1] for H, and to facilitate formal security proofs, a prefix is prepended whenever data is hashed. This prefix is chosen so that every single hash value that is computed uses a unique prefix. In LM-OTS, the prefix includes a 32-byte string I that was randomly generated to identify the Merkle tree associated with the LM-OTS instance, and a 4-byte number q that identifies the tree's leaf that is associated with the LM-OTS signature instance. Other quantities included in LM-OTS prefixes are the index i of a w-bit block within the message hash or checksum, the position j within a Winternitz hash chain, and two-byte constants `0x8080` and `0x8181` used in computing the public key K and message hash, respectively. Additionally, the message M is randomized prior to hashing by prepending a 256-bit random string C, which is included in the signature. The inclusion of C ensures that the message hash need not be collision resistant. Another difference from the Winternitz scheme is that the LM-OTS public key K is derived as the hash of the Winternitz public key; this reduces the size of the OTS public key.

The NIST standard SP 800-208 specifies several parameter sets for LM-OTS. One parameter set uses SHA-256 for H (so $n = 256$), and Winternitz parameter $w = 1$, 2, 4 or 8; the corresponding values of ℓ are 265, 133, 67 and 34.

LMS details. We now describe LMS, which relies on LM-OTS from Alg. 15.2. LMS uses a height-d Merkle hash tree structure whose leaves are associated with LM-OTS public keys; Figure 15.6 depicts a height-3 Merkle tree. The heights permitted in the

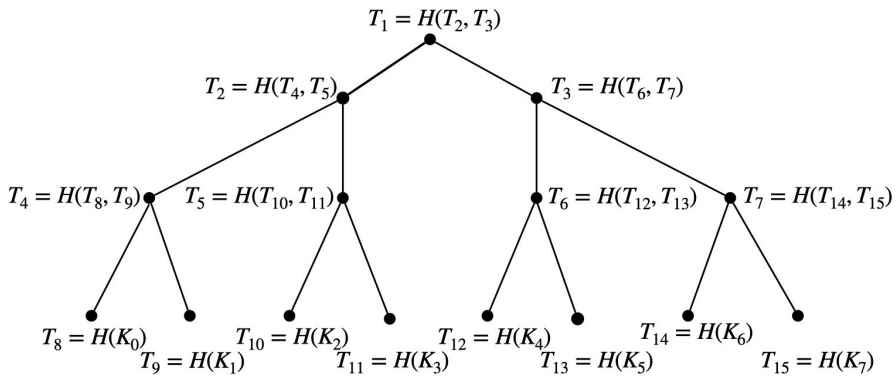

Figure 15.6: A height-3 Merkle tree for LMS.

[1] As explained in Chapter 4, collision resistance is a more stringent security requirement than preimage resistance and second-preimage resistance. Thus, it's always preferable to avoid reliance on collision resistance.

Algorithm 15.2: Leighton-Micali one-time signature scheme (LM-OTS)

Domain parameters. Hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$, Winternitz parameter w, and block lengths ℓ_1, ℓ_2, ℓ (see equation 15.1).

Input. 32-byte Merkle tree identifier I, and 4-byte identifier q of the leaf associated with the LM-OTS key pair.

Key generation. Alice generates a public-private key pair as follows.

for $i = 0$ **to** $\ell - 1$ **do**
    Select $x_i \in_R \{0,1\}^n$ // x_i is a random n-bit string
    tmp $\leftarrow x_i$
    for $j = 0$ **to** $2^w - 2$ // compute the hash chain starting at x_i
     do
        tmp $\leftarrow H(I, q, i, j, \text{tmp})$
    $y_i \leftarrow$ tmp
Compute $K = H(I, q, \texttt{0x8080}, y_0, y_1, \ldots, y_{\ell-1})$.
Alice's private key is $X = (x_0, x_1, \ldots, x_{\ell-1})$; her public key is (I, q, K).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following.

Select $C \in_R \{0,1\}^n$.
Compute $h = H(I, q, \texttt{0x8181}, C, M)$; the w-bit blocks of h are $(h_0, h_1, \ldots, h_{\ell_1-1})$.
Compute csum(h) using equation 15.2.
Obtain the w-bit blocks $(h_{\ell_1}, h_{\ell_1+1}, \ldots, h_{\ell-1})$ of $\tilde{c}(h)$, the binary representation of csum(h) of bitlength $\ell_2 w$
for $i = 0$ **to** $\ell - 1$ **do**
    tmp $\leftarrow x_i$
    for $j = 0$ **to** $h_i - 1$ // compute $s_i = G^{h_i}(x_i)$
     do
        tmp $\leftarrow H(I, q, i, j, \text{tmp})$
    $s_i \leftarrow$ tmp
Alice's signature on M is $(C, s_0, s_1, \ldots, s_{\ell-1})$.

Signature verification. To verify Alice's signature $(C, s_0, \ldots, s_{\ell-1})$ on M, Bob does:

Compute $h = H(I, q, \texttt{0x8181}, C, M)$; the w-bit blocks of h are $(h_0, h_1, \ldots, h_{\ell_1-1})$.
Compute csum(h) and obtain the w-bit blocks $(h_{\ell_1}, h_{\ell_1+1}, \ldots, h_{\ell-1})$ of $\tilde{c}(h)$.
for $i = 0$ **to** $\ell - 1$ **do**
    tmp $\leftarrow s_i$
    for $j = h_i$ **to** $2^w - 2$ // compute $y'_i = G^{2^w-1-h_i}(x_i)$
     do
        tmp $\leftarrow H(I, q, i, j, \text{tmp})$
    $y'_i \leftarrow$ tmp
Compute $K' = H(I, q, \texttt{0x8080}, y'_0, y'_1, \ldots, y'_{\ell-1})$.
if $K' = K$ **then return** (ACCEPT)
else return (REJECT)

NIST SP 800-208 standard are $d = 5$, 10, 15, 20 and 25. The nodes of a tree are numbered sequentially from left to right, and top to bottom. So, the root is assigned number 1. The left child of a node i is assigned number $2i$, and the right child of node i is assigned number $2i + 1$. Thus, the leaves of a height-d tree are numbered 2^d, $2^d + 1, \ldots, 2^{d+1} - 1$. The hash value held in the leaf numbered i is $T_i = H(I, i, \texttt{0x8282}, K_{i-2^d})$, where I is the 32-bit tree identifier, and K_i is the public key for the ith LM-OTS instance. The hash value associated with an internal node numbered i is $T_i = H(I, i, \texttt{0x8383}, T_{2i}, T_{2i+1})$. (For simplicity, Figure 15.6 omits the first few arguments.) The long-term public key associated with the Merkle tree is T_1.

The private keys of the 2^d LM-OTS public keys associated with the leaves of a height-d Merkle tree I are all generated from the same n-bit random seed, denoted SEED. Specifically, for each $0 \le i \le \ell - 1$, the ith component of the private key associated with the leaf numbered q is $x_i = H(I, q, i, \texttt{0xff}, \text{SEED})$.

Signature generation and verification work as outlined on page 9; see also Exercise 15.7.

HSS details. HSS is the hypertree variant of LMS, and should be used if the signer anticipates signing a large number (e.g., 2^{40}) of messages without changing their long-term public key, or if the signer wishes to split the signing workload across two or more cryptographic modules. The NIST standard SP 800-208 permits hypertrees with as many as 8 layers.

15.2.4 SPHINCS+ stateless signature scheme

An LMS signer must exercise extreme caution to avoid reusing any of their OTS private keys since reuse can result in a complete loss of security. To prevent this, the signer can keep track of a counter or key index that increments after each message is signed. This type of data that changes over time is referred to as *state*. However, maintaining state securely in practice can be challenging. For example, if a system crashes or resets and is restored to an earlier state, a private key will likely be reused.

Signature schemes that rely on secure state management for their security are called *stateful*. As mentioned above, LMS is a stateful signature scheme as it requires state to select nodes sequentially. In contrast, the RSA and elliptic curve signature schemes described in Chapters 8 and 9 are *stateless*. Stateful signature schemes are only suitable for applications where state can be reliably managed, e.g., to authenticate software updates wherein signing is infrequent.

SPHINCS+ is a stateless signature scheme standardized by NIST in FIPS 205, where its official name is Stateless Hash-Based Digital Signature Algorithm (SLH-DSA). Similar to LMS, SPHINCS+ employs a hypertree structure, but selects leaf nodes *at random* instead of sequentially, thus avoiding the need to store state information. Recall that the leaves of a hypertree represent key pairs for a one-time signature scheme. The number of leaves is chosen to be exceptionally large, for example 2^{256}. To sign a message, the signer *randomly selects* one of the leaves and signs the message with the associated OTS private

key. A limit is placed on the total number of messages that can be signed, making the probability of OTS private key reuse negligibly small. SPHINCS+ incorporates numerous optimizations to enhance security and efficiency, and reduce public key and signature sizes. A complete description is beyond our scope and goals. Instead, we'll present two main ideas used in SPHINCS+: *few-times signature schemes* and *virtual hypertrees*.

Few-times signature schemes. Unlike a one-time signature scheme, a few-times signature scheme (FTS) can be used a small number of times (e.g., 5–10 times) before security begins to degrade. The core idea of an FTS is that each signature should reveal only a small fraction of the private key components, unlike one-time signature schemes like Lamport's OTS where a single signature reveals half of the private key.

HORS (Hashing to Obtain a Random Subset) is an example of an FTS; see Algorithm 15.3. HORS relies on a preimage resistant hash function $G : \{0,1\}^n \rightarrow \{0,1\}^n$ and a second-preimage resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$. A divisor k of n is chosen, which determines the values $m = n/k$ and $t = 2^m$.

Algorithm 15.3: HORS few-times signature scheme

Domain parameters: Hash functions $G : \{0,1\}^n \rightarrow \{0,1\}^n$ and $H : \{0,1\}^* \rightarrow \{0,1\}^n$, and a divisor k of n (which determines $m = n/k$ and $t = 2^m$).

Key generation. Alice generates a private key X and public key Y as follows.

1 Select $x_i \in_R \{0,1\}^n$ for $0 \leq i \leq t-1$ and set $X = (x_0, x_1, \ldots, x_{t-1})$.
2 Compute $y_i = G(x_i)$ for $0 \leq i \leq t-1$ and set $Y = (y_0, y_1, \ldots, y_{t-1})$.
3 **return** (X, Y).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following.

4 Select $C \in_R \{0,1\}^n$ and compute $h = H(C, M)$.
5 Write $h = (h_0, h_1, \ldots, h_{k-1})$ in base-t.
6 Set $S = (s_0, s_1, \ldots, s_{k-1})$ where $s_j = x_{h_j}$.
7 **return** (C, S).

Signature verification. Bob verifies Alice's signature (C, S) on M as follows.

8 Obtain an authentic copy of Alice's public key Y.
9 Compute $h = H(C, M)$.
10 Write $h = (h_0, h_1, \ldots, h_{k-1})$ in base-t.
11 **for** $j = 0$ **to** $k-1$ **do**
12 If $G(s_j) \neq y_{h_j, j}$ then **return** (REJECT)
13 **return** (ACCEPT).

Alice's private key $X = (x_0, x_1, \ldots, x_{t-1})$ consists of t randomly-selected strings $x_i \in_R \{0,1\}^n$, $0 \leq i \leq t-1$, and her public key consists of their hash values $y_i = G(x_i)$. To sign a message $M \in \{0,1\}^*$, Alice first computes $h = H(C, M) \in \{0,1\}^n$, where $C \in_R \{0,1\}^n$ is a message randomizer. She then forms the base-t representation $(h_0, h_1, \ldots, h_{k-1})$ of h by splitting h into m-bit pieces. Observe that the h_j are not nec-

essarily distinct. Alice's signature on M is $S = (s_0, s_1, \ldots, s_{k-1})$ where $s_j = x_{h_j}$, along with C. Thus, the values h_j determine which private key components to include in the signature. To verify Alice's signature, Bob computes $h = H(C, M)$ and confirms that $G(s_j) = y_{h_j}$ for each $0 \leq j \leq k-1$.

A signature (C, S) exposes at most k of the t components of Alice's private key X, specifically the values x_{h_j}. Suppose now that an adversary has collected c signed messages $(M_i, (C_i, S_i))$. Since the randomizers C_i were chosen by Alice, the adversary has no control over the hash values $h^{(i)} = H(C_i, M_i)$ even if the adversary selected the messages M_i themselves. Assuming H behaves as a random function, the c signed messages reveal to the adversary at most ck *randomly-selected* components of X. Now, if the adversary wishes to forge Alice's signature on any message M^*, she can only do so if each $x_{h_j^*}$ is among the revealed private key components, where $h^* = H(C^*, M^*) = (h_0^*, h_1^*, \ldots, h_{k-1}^*)$ and C^* is selected by the adversary. This happens with probability at most $(ck/t)^k$. Provided that this value is negligible, Alice can securely sign up to c messages.

For example, with parameters $n = 256$, $k = 16$, and $m = 16$ (yielding $t = 2^{16}$), signing $c = 5$ messages limits the adversary's success probability to at most 2^{-154}, which is negliligble. Thus, HORS can securely sign five messages with these parameters.

HORS offers fast signature generation and verification, along with compact signatures. However, the public key Y is large: tn bits, equalling 2.1 Mbytes when $n = 256$ and $k = 16$. HORST (the "T" stands for "with Trees") mitigates this drawback by using a Merkle tree, at the expense of larger signatures. The $t = 2^m$ components y_i of Y are assigned to the leaves of a height-m Merkle tree, with the root R serving as the (small) public key. Alice's signature (C, S) on a message includes the authentication path values for the k leaves labelled y_{h_j} to the root; these components of Y are required for signature verification. Verification of the authentication paths confirms the authenticity of the y_{h_j} values, assuming the verifier already has an authentic copy of R.

Virtual hypertrees. Consider a height-d Merkle tree whose $N = 2^d$ leaves are associated with the public keys for a one-time signature scheme. (Alternatively, one could use a hypertree of total height d.) Since we want the signature scheme to be stateless, messages are signed by selecting a leaf at random and using the associated private key. Suppose that a limit of $S = 2^{40}$ is placed on the number of messages that can be signed. By the birthday paradox, the probability that any OTS private key is reused for two or more messages is at most

$$\binom{S}{2}\frac{1}{N} \approx \frac{S^2}{2N}. \tag{15.3}$$

For example, with $S = 2^{40}$ and $d = 207$, the probability of private key reuse is at most $1/2^{128}$, which is negligible. While this signature scheme is stateless, it's completely impractical because constructing a height-207 Merkle tree would require on the order of 2^{207} operations.

To reduce the tree height, each leaf can be associated with the public key for a few-times signature scheme (such as HORS) instead of an OTS. Messages are still signed using

a randomly selected leaf's private key. Suppose that the FTS allows up to c signatures. We need to check that the probability of any FTS private key being reused more than c times is negligible. By a generalization of the birthday paradox to multicollisions, this probability this is at most

$$\binom{S}{c}\frac{1}{N^{c-1}} \approx \frac{S^c}{c!\,N^{c-1}}. \tag{15.4}$$

For example, with $c = 5$, $S = 2^{40}$ and $d = 80$, the probability that an FTS private key is used more than c times is at most $1/2^{127}$, which is is negligible. However, the time required to construct the height-80 Merkle tree is on the order of 2^{80}, which is still impractical.

SPHINCS+ employs *virtual hypertrees* to address this problem. A virtual hypertree has L layers, each layer consisting of Merkle trees of height d; see Figure 15.7. For example, one might select $L = 7$ and $d = 9$. Each leaf of the Merkle trees in layers 1 through $L-1$ is associated with an OTS public key, whereas each Merkle tree leaf in the Lth layer corresponds to an FTS public key. Each OTS private key is used to sign the root of the tree beneath it, while the FTS private keys are used to sign messages. The root R of the hypertree is the user's long-term public key.

Importantly, all OTS and FTS private keys are derived by applying a pseudorandom function to a single secret value (SEED) and a unique index that identifies the corresponding public key. This index might include the layer number, the tree identifier within that layer, and the public key's position within that tree. Once the hypertree's parameters and the SEED are fixed, the entire hypertree is deterministically defined—including all Merkle tree hash values and the OTS/FTS key pairs. Even though constructing the entire hypertree might be computationally infeasible (this is the sense in which the hypertree is "virtual"), any individual Merkle tree within it can be efficiently constructed from the SEED. This is done by first computing the relevant OTS or FTS public keys for the leaves of the tree, and then deriving the hash values for the tree's nodes.

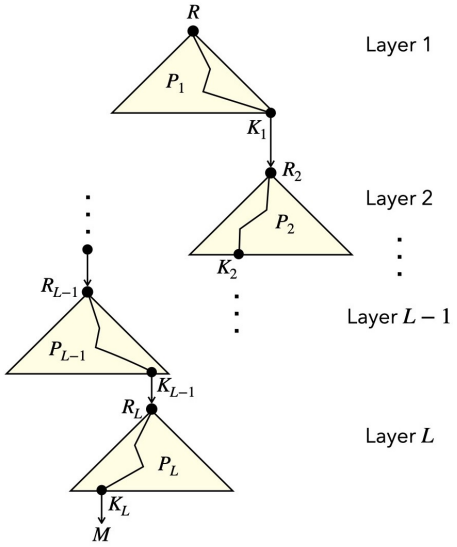

Figure 15.7: A virtual hypertree.

Signature generation using hypertrees. To sign a message M, Alice begins by selecting a random FTS public key K_L (as shown in Figure 15.7). This could be accomplished by hashing the message and SEED, and using the resulting hash value to identify a leaf index in the hypertree. She then signs the message using the private key corresponding

to K_L. Next, Alice reconstructs the root R_L of the Merkle tree containing K_L and derives the authentication path P_L for this key. Using the layer above, she derives the OTS public key K_{L-1} (using SEED as described above), signs R_L with the private key corresponding to K_{L-1}, reconstructs the root R_{L-1} of the Merkle tree containing K_{L-1}, and derives the authentication path P_{L-1} for K_{L-1}. This process is continued upward through each layer until reaching the hypertree's topmost root R. The overall signature on M is comprised of the initial FTS signature, all intermediate OTS signatures, the public keys $K_1, \ldots, K_L$, and the authentication paths $P_1, \ldots, P_L$.

Signature verification using hypertrees. To verify a signed message, Bob first validates the authentication path P_1 from K_1 to its Merkle root R. He proceeds to verify the OTS signature on R_2 using K_1 and validates the authentication path P_2 from K_2 to R_2. This verification process continues layer by layer down the hypertree, with each step authenticating both the OTS signature and associated Merkle tree authentication path. When Bob successfully verifies all authentication paths and signatures down to the FTS public key K_L, he is confident of the authenticity of K_L that he was sent. Finally, he verifies the FTS signature using K_L.

15.3 Kyber-KEM

Kyber-KEM is a quantum-safe key encapsulation mechanism (KEM) standardized by NIST in FIPS 203. It is designed to offer strong security guarantees against attacks from both classical and quantum computers. Kyber-KEM is based on the Kyber public-key encryption scheme (Kyber-PKE), described in §15.3.4. The security of both Kyber-PKE and Kyber-KEM relies on the hardness of the Decisional Module Learning With Errors Problem (D-MLWE), which will be described in §15.3.3, following the introduction of the Learning With Errors (LWE) problem.

15.3.1 Learning with errors problem

The Learning With Errors (LWE) problem was introduced by Oded Regev in 2005. Its hardness has served as the security foundation for a remarkably broad range of cryptographic constructions, including quantum-safe key encapsulation and signature schemes, and fully-homomorphic encryption.

Notation. q is a prime; $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$ is the set of integers modulo q; $\mathbb{Z}_q^n$ is the set of length-n (column) vectors whose components are in $\mathbb{Z}_q$; $\mathbb{Z}_q^{m \times n}$ is the set of $m \times n$ matrices whose entries are in $\mathbb{Z}_q$; and $[-B, B]^m$ denotes the set of all length-m (column) vectors whose components are integers in the interval $[-B, B]$.

Definition 15.3 The *Learning With Errors* problem LWE(m, n, q, B) is the following. Let $A \in_R \mathbb{Z}_q^{m \times n}$, $s \in_R \mathbb{Z}_q^n$, $e \in_R [-B, B]^m$, and $b = As + e \pmod q \in \mathbb{Z}_q^m$. Given a problem instance (A, b), find s; see Figure 15.8. Here, the error bound B is a positive integer that is significantly smaller than $q/2$.

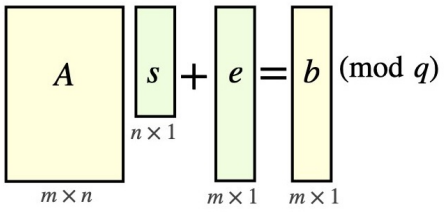

Figure 15.8: The LWE(m, n, q, B) problem.

Observe that if $B = 0$, whence $e = 0$, then the problem is to solve the system of linear equations $As = b \pmod q$ for s. In this case, the set of all solutions can be efficiently determined using Gaussian elimination over $\mathbb{Z}_q$. Hardness of LWE arises because of the introduction of the noise e. We henceforth assume that $B \neq 0$.

The statement of the LWE problem implicitly assumes that the LWE solution s is unique. In fact, the solution is expected to be unique with overwhelming probability provided that the number m of rows of A is sufficiently greater than the number n of columns of A.

To support this claim, observe that multiplying A by a vector $s \in \mathbb{Z}_q^n$ results in a vector $t = As \pmod q$ in $\mathbb{Z}_q^m$, a vector space with q^m elements. For each $s \in \mathbb{Z}_q^n$, the sphere centered at $t = As \pmod q$ is defined as $T_s = \{t + e \pmod q \mid e \in [-B, B]^m\}$; note that the number of vectors in each sphere is $|T_s| = (2B + 1)^m$. Uniqueness of the LWE solution is guaranteed only if no two of the q^n spheres overlap (see Figure 15.9). Now, if the centers of these spheres are selected at random, and if $m \gg n$ and $B \ll q/2$,

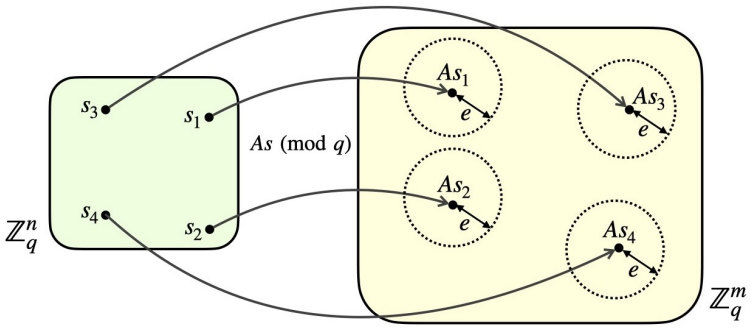

Figure 15.9: Uniqueness of the LWE solution.

then the q^n spheres will be relatively small in both number and size, meaning that very few, if any, will overlap. As a result, the probability of a non-unique LWE solution will be very low. Therefore, we will assume that $m \gg n$ and that the LWE solution is unique.

Example 15.4 (*LWE instance*) Let $m = 5$, $n = 3$, $q = 43$, and $B = 2$. Consider the

LWE(m, n, q, B) instance

$$A = \begin{bmatrix} 17 & 3 & 27 \\ 12 & 41 & 7 \\ 6 & 23 & 40 \\ 35 & 6 & 22 \\ 21 & 0 & 14 \end{bmatrix} \in \mathbb{Z}_{43}^{5\times 3}, \qquad b = \begin{bmatrix} 35 \\ 32 \\ 10 \\ 1 \\ 23 \end{bmatrix} \in \mathbb{Z}_{43}^{5}.$$

The LWE challenge is to find $s \in \mathbb{Z}_{43}^3$ and $e \in [-2, 2]^3$ such that $As + e = b \pmod{43}$. It turns out that this LWE instance has two solutions: $s = (2, 34, 17)$, $e = (-1, 0, -2, -2, 1)$; and $s = (25, 8, 7)$, $e = (-1, 0, -2, -1, 2)$.

Decisional LWE. In the decisional variant of LWE, the challenge is to distinguish an LWE instance (A, b) from a random instance (A, r) where $r \in \mathbb{Z}_q^m$ is randomly selected. In the latter case, one expects that $As + e = c \pmod{q}$ does not have an LWE solution. Essentially, the decisional problem is to decide whether a solution exists.

Definition 15.5 The *Decisional Learning With Errors* problem DLWE(m, n, q, B) is the following. Let $A \in_R \mathbb{Z}_q^{m\times n}$, $s \in_R \mathbb{Z}_q^n$, $e \in_R [-B, B]^m$, $r \in_R \mathbb{Z}_q^m$, and let $b = As + e \pmod{q}$. With equal probability $\frac{1}{2}$, set $c = b$ or $c = r$. Given a problem instance (A, c), decide (with success probability significantly greater than $\frac{1}{2}$) whether $c = b$ or $c = r$.

Theorem 15.6 LWE and DLWE are computationally equivalent.

Proof: We'll first prove that LWE reduces to DLWE (denoted LWE $\leq$ DLWE) and then prove that DLWE reduces to LWE (denoted DLWE $\leq$ LWE).

(LWE $\leq$ DLWE) Let (A, b) be an LWE instance (where $b = As + e$). Assume we have a DLWE-solver to test guesses for the coordinates of s, one at a time, beginning with the first coordinate s_1. We know there is a solution s, and proceed to find it. This procedure may need to be repeated up to qn times before s is determined.

Let $d \in \mathbb{Z}_q$. To test whether $s_1 = d$, select $\Delta \in_R \mathbb{Z}_q^m$. Let A' be the matrix obtained by adding Δ to the first column of A, and let $b' = b + d\Delta$.

Now, if $s_1 = d$, then $b' = A's + e$ so (A', b') is a valid LWE instance. On the other hand, if $s_1 \neq d$, then $b' = A's + e + (d - s_1)\Delta$. Since $d - s_1$ is nonzero, and Δ is uniformly random and independent of A', s and e, it follows that b' is uniformly random and independent of A'. Thus, (A', b') is a valid DLWE instance, whence the DLWE solver with input (A', b') will inform us whether or not $s_1 = d$. We eventually find s_1 and the other s_i's similarly.

(DLWE $\leq$ LWE) Let (A, c) be a DLWE instance. Assume that we have an LWE-solver to efficiently solve the DLWE instance.

Now, if $c = b$, then (A, c) is an LWE instance and so one expects that $As + e = c \pmod{q}$ has a unique LWE solution (s, e). And, if $c = r$, as noted earlier one expects that $As + e = c \pmod{q}$ does not have an LWE solution. So, the LWE solver is run with input (A, c). If a valid LWE solution is returned, then we conclude that $c = b$. If the LWE solver terminates without a valid LWE solution, or fails to terminate, then we conclude that $c = r$. □

Short-secret LWE. In the short-secret variant ss-LWE(m, n, q, B) of LWE, and the short-secret variant ss-DLWE(m, n, q, B) of DLWE, the secret vector s is selected uniformly at random from $[-B, B]^n$ instead of from $\mathbb{Z}_q^n$. LWE and ss-LWE are computationally equivalent, as are ss-LWE and ss-DLWE (Exercises 15.13, 15.14 and 15.15).

Hardness of LWE. It seems reasonable to conjecture that LWE is hard in the *worst case* (and this is the current belief). However, since an LWE instance is generated by selecting A, s and e uniformly at random, in cryptographic applications that rely on LWE hardness we also need the assurance that LWE is hard *on average*, meaning that no efficient algorithm can solve the problem with non-negligible success probability.

In his 2005 paper, Oded Regev proved that if approx-SIVP, a natural computational problem involving lattices, is hard in the worst case against quantum attacks (and classical attacks), then LWE is hard on average. To explain this further, an n-dimensional lattice L with basis $\{v_1, v_2, \ldots, v_n\}$ consists of all integer linear combinations of n linearly independent vectors $v_1, v_2, \ldots, v_n$ in $\mathbb{R}^n$. The theory of lattices has a vast body of literature, with applications spanning mathematics, computer science, and physics. Two key computational problems in lattice theory are the shortest vector problem (SVP), where the goal is to find a nonzero lattice vector of smallest (Euclidean) length, and the short independent vectors problem (SIVP), where the goal is to find n linearly independent lattice vectors such that the longest vector is as short as possible. Both SVP and SIVP are widely believed to be hard in the worst case, as are their approximate versions (approx-SVP and approx-SIVP), where the objective is to find a nonzero lattice vector or a set of n linearly independent lattice vectors within a small factor of the shortest possible.

Since the assumption that approx-SIVP is quantumly hard in the worst case is widely accepted, Regev's proof, which is an example of a *worst-case to average-case reduction*, offers a provable (though conditional) guarantee that LWE is hard on average. However, Regev's reduction is very inefficient, offering only an asymptotic hardness guarantee for LWE as the parameters tend to infinity. As a result, it is unclear what guarantees Regev's proof provides for the hardness of LWE for practical parameter sizes.

In practice, the best method known for solving LWE is to reformulate the problem as an instance of SVP and then apply the fastest known SVP solvers, all of which have fully-exponential running times. The performance of these SVP solvers helps determine the appropriate LWE parameters for use in real-world LWE-based cryptosystems. This background explains why these cryptosystems are always called *lattice-based* even though knowledge of lattices is not required to implement the cryptographic mechanisms.

15.3.2 The polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$

We next introduce polynomial rings, which will be used in §15.3.3 to define a module variant of LWE. Let q be a prime and n a positive integer. The *polynomial ring* $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ is composed of the polynomials in $\mathbb{Z}_q[x]$ of degree less than n, with multiplication performed using the *reduction polynomial* $x^n + 1$.

To multiply two polynomials $f(x), g(x) \in R_q$, one (i) multiplies them using ordinary polynomial multiplication in R_q to obtain a polynomial $h(x) \in \mathbb{Z}_q[x]$ of degree $\leq 2n - 2$;

and then (ii) divides $h(x)$ by $x^n + 1$ to get a remainder polynomial $r(x) \in \mathbb{Z}_q[x]$ of degree at most $n - 1$. The product of $f(x)$ and $g(x)$ in R_q is the remainder polynomial $r(x)$.

Example 15.7 (*polynomial ring*) Let $q = 43$ and $n = 4$. Then $R_q = \mathbb{Z}_{43}[x]/(x^4 + 1)$ is composed of the polynomials in $\mathbb{Z}_{43}[x]$ of degree at most 3.

Let $f(x) = 33 + 41x + 12x^3 \in R_q$ and $g(x) = 2 + 17x + 39x^2 + 14x^3 \in R_q$. Then their sum and difference in R_q are $f(x) + g(x) = 35 + 15x + 39x^2 + 26x^3$ and $f(x) - g(x) = 31 + 24x + 4x^2 + 41x^3$. The product of f and g in $\mathbb{Z}_{43}[x]$ is the degree-6 polynomial $h(x) = 23 + 41x + 6x^2 + 21x^3 + 4x^4 + 38x^5 + 39x^6$. The division of $h(x)$ by $x^4 + 1$ can be accomplished by replacing x^4 by -1, x^5 by $-x$, and x^6 by $-x^2$, and then simplifying. We obtain $r(x) = 23 + 41x + 6x^2 + 21x^3 - 4 - 38x - 39x^2 = 19 + 3x + 10x^2 + 21x^3$. Thus, the product of f and g in R_q is $f(x) \times g(x) = r(x) = 19 + 3x + 10x^2 + 21x^3$.

Polynomials as vectors. Let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. A polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \in R_q$ can be represented by its length-n vector of coefficients $f = (a_0, a_1, a_2, \ldots, a_{n-1})$.

Example 15.8 (*representing polynomials as vectors*) Let $q = 31$ and $n = 5$, so $R_q = \mathbb{Z}_{31}[x]/(x^5 + 1)$. Let $f(x) = 12 + 3x + 4x^3 + 30x^4 \in R_q$ and $g(x) = 7 + 13x + 22x^2 + 27x^3 + 28x^4 \in R_q$. The vector representations of $f(x)$ and $g(x)$ are $f = (12, 3, 0, 4, 30)$ and $g = (7, 13, 22, 27, 28)$. You can check that the vector representations of the sum, difference, and products of $f(x)$ and $g(x)$ in R_q are $f + g = (19, 16, 22, 0, 27)$, $f - g = (5, 21, 9, 8, 2)$, and $f \times g = (18, 29, 1, 12, 28)$.

Polynomial multiplication via matrices. Polynomial multiplication in R_q can be expressed as a matrix-vector product. Let $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \in R_q$ and $b(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1} \in R_q$, and let $c(x) = a(x) \times b(x)$ in R_q. Then

$$\begin{aligned} c(x) &= a(x)(b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}) \\ &= b_0a(x) + b_1x \cdot a(x) + b_2x^2 \cdot a(x) + \cdots + b_{n-1}x^{n-1} \cdot a(x). \end{aligned} \tag{15.5}$$

Now,

$$\begin{aligned} x \cdot a(x) &= a_0x + a_1x^2 + \cdots + a_{n-2}x^{n-1} + a_{n-1}x^n \\ &= -a_{n-1} + a_0x + a_1x^2 + \cdots + a_{n-2}x^{n-1} \pmod{x^n + 1} \\ &\leftrightarrow (-a_{n-1}, a_0, a_1, \ldots, a_{n-2}) \end{aligned}$$

since $x^n = -1 \pmod{x^n + 1}$. Thus, the vector representation of $x \cdot a(x)$ is the right cyclic shift of the vector representation of $a(x)$, with the cycled element negated. It follows from (15.5) that if $c(x) = a(x) \times b(x)$ in R_q, then

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} a_0 & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \\ a_1 & a_0 & -a_{n-1} & \cdots & -a_2 \\ a_2 & a_1 & a_0 & \cdots & -a_3 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix}.$$

The matrix above is an *anti-circulant matrix*, denoted $A = \overline{\text{circ}}(a)$.

Example 15.9 (*matrix representation of R_q multiplication*) Let $q = 17$ and $n = 5$, so $R_q = \mathbb{Z}_{17}[x]/(x^5+1)$. Let $a(x) = 11+2x+9x^3+13x^4$, $b(x) = 5+3x+14x^2+7x^3+15x^4 \in R_q$. Then the vector representation of $c(x) = a(x)b(x)$ is

$$c = \begin{bmatrix} 11 & 4 & 8 & 0 & 15 \\ 2 & 11 & 4 & 8 & 0 \\ 0 & 2 & 11 & 4 & 8 \\ 9 & 0 & 2 & 11 & 4 \\ 13 & 9 & 0 & 2 & 11 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ 14 \\ 7 \\ 15 \end{bmatrix} = \begin{bmatrix} 13 \\ 2 \\ 2 \\ 6 \\ 16 \end{bmatrix}$$

Hence, $a(x)b(x) = 13 + 2x + 2x^2 + 6x^3 + 16x^4$ in R_q.

We'll next define a notion of "size" for integers in $\mathbb{Z}_q$ and polynomials in R_q.

Definition 15.10 Let $q \geq 3$ be an odd integer, and let $r \in \mathbb{Z}_q$. Define the *symmetric mod q* of r to be

$$r \bmod q = \begin{cases} r, & \text{if } r \leq (q-1)/2 \\ r-q, & \text{if } r > (q-1)/2. \end{cases}$$

Note that $-(q-1)/2 \leq r \bmod q \leq (q-1)/2$. For example, if $q = 17$ and $r \in \mathbb{Z}_{17}$ then $-8 \leq r \bmod 17 \leq 8$. We have $6 \bmod 17 = 6$ and $13 \bmod 17 = -4$.

Definition 15.11 Let $q \geq 3$ be an odd integer. The *size* of an integer $r \in \mathbb{Z}_q$, also called the *infinity norm* of r, is $\|r\|_\infty = |r \bmod q|$ where $|\cdot|$ denotes absolute value.

Note that $0 \leq \|r\|_\infty \leq (q-1)/2$. For example, if $q = 17$ then $0 \leq \|r\|_\infty \leq 8$. We have $\|6\|_\infty = 6$ and $\|13\|_\infty = 4$.

Definition 15.12 The *size* of a polynomial $f(x) = f_0 + f_1x + f_2x^2 + \cdots + f_{n-1}x^{n-1} \in R_q$ is $\|f\|_\infty = \max_i \|f_i\|_\infty$, also called the *infinity norm* of $f(x)$.

Definition 15.13 Let η be a positive integer that is small compared to $q/2$. The set of *small polynomials* in R_q is $S_\eta = \{f \in R_q \mid \|f\|_\infty \leq \eta\}$.

The product of small polynomials is also (relatively) small, as shown next.

Theorem 15.14 Fix $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Let η_1 and η_2 be positive integers such that $n\eta_1\eta_2 < q/2$. Let $f \in S_{\eta_1}$ and $g \in S_{\eta_2}$. Then $fg \in S_{n\eta_1\eta_2}$.

Proof: Consider $f(x) = f_0 + f_1x + \cdots + f_{n-1}x^{n-1} \in S_{\eta_1}$ and $g(x) = g_0 + g_1x + \cdots + g_{n-1}x^{n-1} \in S_{\eta_2}$. Let $h(x) = h_0 + h_1x + \cdots + h_{n-1}x^{n-1}$ be the product of $f(x)$ and $g(x)$ in R_q. By multiplying together $f(x)$ and $g(x)$, then replacing x^{n+j} by $-x^j$ for $0 \leq j \leq n-2$, and then simplifying, it follows that the coefficient h_i of x^i in $h(x)$ for each $i \in [0, n-1]$ is:

$$h_i = f_0g_i + f_1g_{i-1} + \cdots + f_ig_0 - f_{i+1}g_{n-1} - f_{i+2}g_{n-2} - \cdots - f_{n-2}g_{i+2} - f_{n-1}g_{i+1}.$$

Hence, $\|h_i\|_\infty \leq n\eta_1\eta_2$, whence $\|h\|_\infty \leq n\eta_1\eta_2$. □

Example 15.15 (*small polynomials*) Let $q = 379$ and $n = 5$ so $R_q = \mathbb{Z}_{379}[x]/(x^5 + 1)$. Let $f(x) = 378 + x^2 + 2x^3 + 377x^4 \in R_q$. The mods q representation of f is $f(x) = -1 + x^2 + 2x^3 - 2x^4$ so $f \in S_2$ is a small polynomial (with respect to $\eta_1 = 2$). Similarly, let $g(x) = 3 + 376x + 2x^2 + 378x^3 + 377x^4 \in S_3$ be a small polynomial (with respect to $\eta_2 = 3$). Then the product of f and g in R_q is $h(x) = 367 + 11x + 3x^2 + 371x^4$ which is in S_{12} since its mods q representation is $h(x) = -12 + 11x + 3x^2 - 8x^4$. Note here that $12 \leq n\eta_1\eta_2 = 30$.

15.3.3 Module-LWE (MLWE)

Having introduced LWE and DLWE, we now extend these into counterparts Module Learning With Errors (MLWE) and Decisional Module Learning With Errors (D-MLWE) by replacing integers modulo q with polynomials in the ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$. Kyber's security is based on the hardness of D-MLWE, which is closely related to the hardness of MLWE.

A *module* extends the notion of a vector space by substituting the vector space's field of scalars with a ring. The modules of interest here are R_q^k, which consists of length-k vectors of polynomials in R_q. Addition and subtraction of elements in R_q^k are performed component-wise, ensuring that the result remains within R_q^k. The *inner product* of two vectors $a = (a_1, a_2, \ldots, a_k)$ and $b = (b_1, b_2, \ldots, b_k)$ in R_q^k is defined to be

$$a^T \cdot b = a_1b_1 + a_2b_2 + \cdots + a_kb_k,$$

which is a polynomial in R_q.

Example 15.16 (*module*) Let $q = 31$, $n = 5$ (so $R_q = \mathbb{Z}_{31}[x]/(x^5 + 1)$) and $k = 3$. Let $a = (a_0, a_1, a_2)^T$ and $b = (b_0, b_1, b_2)^T$ be:

$$a = \begin{bmatrix} 12 + 7x + x^3 + 29x^4 \\ 3 + 17x + 22x^2 + 9x^3 + 16x^4 \\ 14 + 8x + 29x^2 + 7x^3 + 4x^4 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} 25 + 12x + 5x^2 + 23x^3 + 18x^4 \\ 3 + 29x + x^2 + 4x^3 + 7x^4 \\ 3 + 24x + 14x^2 + 4x^3 + 7x^4 \end{bmatrix} \in R_q^3.$$

Then

$$a + b = \begin{bmatrix} 6 + 19x + 5x^2 + 24x^3 + 16x^4 \\ 6 + 15x + 23x^2 + 13x^3 + 23x^4 \\ 17 + x + 12x^2 + 11x^3 + 11x^4 \end{bmatrix}, \quad a - b = \begin{bmatrix} 18 + 26x + 26x^2 + 9x^3 + 11x^4 \\ 19x + 21x^2 + 5x^3 + 9x^4 \\ 11 + 15x + 15x^2 + 3x^3 + 28x^4 \end{bmatrix},$$

and

$$a^T \cdot b = a_1b_1 + a_2b_2 + a_3b_3 = 4 + x + 25x^2 + 13x^3 + 18x^4.$$

Definition 15.17 The *Module Learning With Errors* problem MLWE(k, ℓ, n, q, B) is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R R_q^\ell$ and $e \in_R S_B^k$. Here, $k > \ell$ and $B \ll q/2$. Given A and $t = As + e \in R_q^k$, find s.

The columns $A_1, A_2, \ldots, A_\ell$ of A belong to the module R_q^k, as per a and b in Example 15.16. MLWE asks to express the module element t, up to a small error e, as a linear combination of the A_i, where the coefficients of the linear combination are polynomials.

Example 15.18 (*MLWE instance*) Let $k = 3$, $\ell = 2$, $n = 4$, $q = 37$ (so $R_q = \mathbb{Z}_{37}[x]/(x^4 + 1)$), and $B = 1$. Randomly select

$$A = \begin{bmatrix} 33 + 34x^2 + 8x^3 & 21 + 10x + 19x^2 + 18x^3 \\ 3 + 13x + 19x^2 + 18x^3 & 34 + 19x + 15x^2 + 4x^3 \\ 16 + 6x + 5x^2 + 7x^3 & 24 + 6x + 35x^2 + 31x^3 \end{bmatrix} \in R_q^{3\times 2},$$

$$s = \begin{bmatrix} 32 + 15x + 16x^2 + 3x^3 \\ 10 + 28x + 6x^2 + 2x^3 \end{bmatrix} \in R_q^2, \quad \text{and } e = \begin{bmatrix} 1 - x \\ 1 - x^2 - x^3 \\ -1 + x - x^3 \end{bmatrix} \in S_1^3,$$

and define t to be

$$t = As + e = \begin{bmatrix} 2 + 29x + 6x^2 + 14x^3 \\ 28 + 20x + 9x^2 + 17x^3 \\ 13 + 5x + 12x^2 + 3x^3 \end{bmatrix} \in R_q^3.$$

Given (A, t), the MLWE challenge is to find $s \in R_q^2$ and $e \in S_1^3$ such that $As + e = t$.

Recall that polynomial multiplication in R_q can be expressed as a matrix-vector product. Consequently, an equivalent formulation of this MLWE instance is to solve $\overline{A}\overline{s}+\overline{e} = \overline{t} \pmod{37}$ for $s \in \mathbb{Z}_{37}^8$ and $e \in [-1, 1]^{12}$, where

$$\overline{A} = \left[\begin{array}{cccc|cccc} 33 & 29 & 3 & 0 & 21 & 19 & 18 & 27 \\ 0 & 33 & 29 & 3 & 10 & 21 & 19 & 18 \\ 34 & 0 & 33 & 29 & 19 & 10 & 21 & 19 \\ 8 & 34 & 0 & 33 & 18 & 19 & 10 & 21 \\ \hline 3 & 19 & 18 & 24 & 34 & 33 & 22 & 18 \\ 13 & 3 & 19 & 18 & 19 & 34 & 33 & 22 \\ 19 & 13 & 3 & 19 & 15 & 19 & 34 & 33 \\ 18 & 19 & 13 & 3 & 4 & 15 & 19 & 34 \\ \hline 16 & 30 & 32 & 31 & 24 & 6 & 2 & 31 \\ 6 & 16 & 30 & 32 & 6 & 24 & 6 & 2 \\ 5 & 6 & 16 & 30 & 35 & 6 & 24 & 6 \\ 7 & 5 & 6 & 16 & 31 & 35 & 6 & 24 \end{array}\right]_{12\times 8} \quad \text{and} \quad \overline{t} = \left[\begin{array}{c} 2 \\ 29 \\ 6 \\ 14 \\ \hline 28 \\ 20 \\ 9 \\ 17 \\ \hline 13 \\ 5 \\ 12 \\ 3 \end{array}\right]_{12\times 1}.$$

The 4×4 blocks in $\overline{A}$ are the anti-circulant matrices corresponding to the polynomials in A, whereas the 4×1 blocks in $\overline{t}$ and the vector representations of the polynomials in t.

As it turns out, there are two MLWE solutions:

$$\overline{s} = (17, 4, 36, 30, 35, 31, 13, 12), \quad \overline{e} = (-1, 0, 1, 0, -1, -1, 1, 0, 0, -1, -1, -1);$$

and

$$\overline{s} = (32, 15, 16, 3, 10, 28, 6, 2), \quad \overline{e} = (1, -1, 0, 0, 1, 0, -1, -1, -1, 1, 0, -1).$$

The polynomial form of the first solution is $s = (17 + 4x + 36x^2 + 30x^3,\ 35 + 31x + 13x^2 + 12x^3)$, $e = (-1 + x^2,\ -1 - x + x^2,\ -x - x^2 - x^3)$. The second solution corresponds to the s and e values that were selected above in the formulation of this MLWE instance.

Definition 15.19 The *Decisional Learning With Errors* problem D-MLWE(k, ℓ, n, q, B) is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R R_q^{\ell}$, $e \in_R S_B^k$, $r \in_R R_q^k$, and let $t = As + e \in R_q^k$. Let $z = t$ with probability $\frac{1}{2}$ and $z = r$ with probability $\frac{1}{2}$. Given A and z, decide (with success probability significantly greater than $\frac{1}{2}$) whether $z = t$ or $z = r$.

In the short-secret variants of MLWE and D-MLWE, the secret polynomial vector s of is also chosen to have small coefficients.

Definition 15.20 The *short-secret Module Learning With Errors* problem ss-MLWE$(k, \ell, n, q, \eta_1, \eta_2)$ is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R S_{\eta_1}^{\ell}$ and $e \in_R S_{\eta_2}^k$. Here, $k > \ell$ and $\eta_1, \eta_2 \ll q/2$. Given A and $t = As + e \in R_q^k$, find s.

Definition 15.21 The *short-secret Decisional Learning With Errors* problem ss-D-MLWE$(k, \ell, n, q, \eta_1, \eta_2)$ is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R S_{\eta_1}^{\ell}$, $e \in_R S_{\eta_2}^k$, $r \in_R R_q^k$, and let $t = As + e \in R_q^k$. Let $z = t$ with probability $\frac{1}{2}$ and $z = r$ with probability $\frac{1}{2}$. Given A and z, decide (with success probability significantly greater than $\frac{1}{2}$) whether $z = t$ or $z = r$.

Hardness of MLWE. No attacks are known on MLWE (or its decisional and short-secret variants) that are any faster than the fastest attacks known on LWE. In other words, no attacks on MLWE are known that exploit the special structure of the matrix A, namely that it is composed of blocks of anti-circulant matrices.

15.3.4 The Kyber public-key encryption scheme

Kyber-PKE is a quantum-safe public-key encryption scheme (PKE) based on the MLWE problem. Kyber-PKE was standardized by NIST in FIPS 203, where its official name is K-PKE. Kyber-PKE is designed for use as a component in the Kyber key encapsulation mechanism (§15.3.5). Indeed, while Kyber-PKE is secure against chosen-plaintext attacks (Theorem 15.24), it does not provide security against chosen-ciphertext attacks (Exercise 15.22) and, therefore, should not be used as a standalone public-key encryption scheme.

Notation. $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, $S_\eta = \{f \in R_q \mid \|f\|_\infty \leq \eta\}$, $\lceil x \rfloor$ denotes the closest integer to x with ties broken upwards. The plaintext space is $\{0, 1\}^n$. A plaintext $m = (m_0, m_1, \ldots, m_{n-1}) \in \{0, 1\}^n$ is associated with the polynomial $m(x) = m_0 + m_1 x + \cdots + m_{n-1}x^{n-1} \in R_q$.

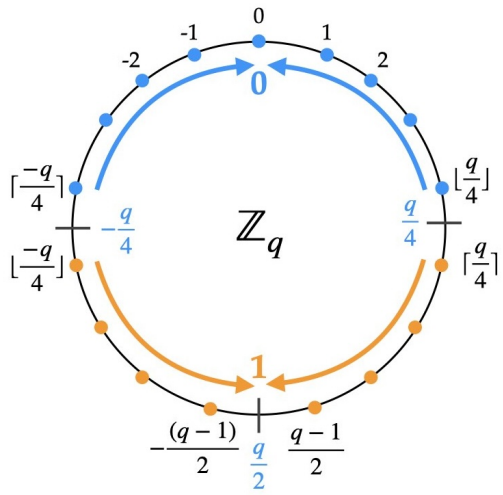

Figure 15.10: Rounding used in Kyber.

Rounding. Kyber uses a form of *rounding*. Let q be an odd prime, and $x \in [0, q-1]$. Recall that $x \bmod q \in [-(q-1)/2, (q-1)/2]$. The rounding function, depicted in Figure 15.10, is:

$$\text{Round}_q(x) = \begin{cases} 0, & \text{if } -q/4 < x \bmod q < q/4, \\ 1, & \text{otherwise.} \end{cases}$$

For example, if $q = 3329$ then $\text{Round}_q(x) = 0$ if $0 \le x \le 832$ or $2497 \le x \le 3328$, and $\text{Round}_q(x) = 1$ if $833 \le x \le 2496$. The rounding function can be extended to polynomials in R_q by applying Round_q to each coefficient of the polynomial. For example, if $q = 3329$ then $\text{Round}_q(3000 + 1500x + 2010x^2 + 37x^3) = x + x^2$.

The Kyber public-key encryption scheme (without ciphertext compression) is presented in Alg. 15.4. The three Kyber parameter sets specified in FIPS 203 are listed in Table 15.1.

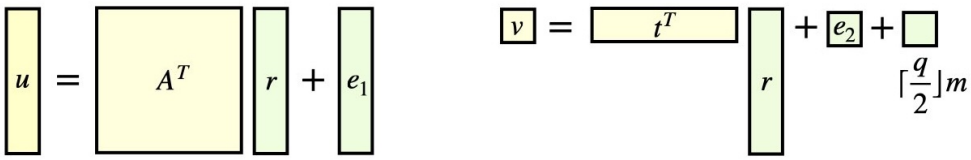

Figure 15.11: Kyber encryption.

Algorithm 15.4: Kyber public-key enc. scheme (without ciphertext compression)

Domain parameters. q, n, k, η_1, η_2.

Key generation. Bob does the following:
1 Select $A \in R_q^{k\times k}$, $s \in_R S_{\eta_1}^k$, and $e \in_R S_{\eta_2}^k$.
2 Compute $t = As + e$.
3 Bob's encryption (public) key is (A, t); her decryption (private) key is s.

Encryption. To encrypt a message $m \in \{0,1\}^n$ for Bob, Alice does the following:
4 Obtain an authentic copy of Bob's encryption key (A, t).
5 Select $r \in_R S_{\eta_1}^k$, $e_1 \in_R S_{\eta_2}^k$, and $e_2 \in_R S_{\eta_2}$.
6 Compute $u = A^T r + e_1$ and $v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m$ (see Figure 15.11).
7 Output $c = (u, v)$.

Decryption. To decrypt $c = (u, v)$, Bob does:
8 Compute $m = \text{Round}_q(v - s^T u)$.

Table 15.1: FIPS 203 parameter sets for Kyber-PKE and Kyber-KEM.

Parameter	Description	ML-KEM-512	ML-KEM-768	ML-KEM-1024
q	prime modulus	3329	3329	3329
n	$R_q = \mathbb{Z}_q[x]/(x^n+1)$	256	256	256
k	dimensions of A	2	3	4
η_1	coefficient bound for s	3	2	2
η_2	coefficient bound for e	2	2	2
(d_u, d_v)	compression parameters	(10,4)	(10,4)	(11,5)

Example 15.22 (*Kyber-PKE*)
Domain parameters. $q = 173$, $n = 4$, $k = 3$, $\eta_1 = 2$, and $\eta_2 = 2$.
Key generation. Bob selects

$$A = \begin{bmatrix} 1+127x+14x^2+86x^3 & 154+123x+120x^2+114x^3 & 126+128x+121x^2+4x^3 \\ 69+35x+86x^2+145x^3 & 13+38x+147x^2+55x^3 & 157+131x+103x^2+147x^3 \\ 61+172x+103x^2+25x^3 & 30+114x+160x^2+133x^3 & 3+104x+114x^2+39x^3 \end{bmatrix},$$

$$s = \begin{bmatrix} -1+x-x^3 \\ -x-2x^2-x^3 \\ 2+2x+x^2+x^3 \end{bmatrix}, \text{ and } e = \begin{bmatrix} 1-x-2x^2-x^3 \\ 1-2x^2-2x^3 \\ 2x-x^2+2x^3 \end{bmatrix},$$

and computes

$$t = As + e = \begin{bmatrix} 167+118x+154x^2+83x^3 \\ 168+171x+42x^2+76x^3 \\ 17+105x+106x^2+14x^3 \end{bmatrix}.$$

Bob's encryption key is (A, t), and her decryption key is s.

ENCRYPTION. To encrypt the plaintext message $m = 1010 \leftrightarrow 1 + x^2$ for Bob, Alice selects

$$r = \begin{bmatrix} 1 + x - 2x^2 + x^3 \\ 2 - 2x - 2x^2 \\ 2 - x + 2x^2 \end{bmatrix}, \quad e_1 = \begin{bmatrix} 1 + x - 2x^2 \\ -1 + x + x^2 + x^3 \\ -1 - 2x + 2x^2 + 2x^3 \end{bmatrix}, \quad \text{and } e_2 = -1 + x^2$$

and computes

$$u = A^T r + e_1 = \begin{bmatrix} 12 + 50x + 171x^2 + 13x^3 \\ 113 + 132x + 107x^2 + 44x^3 \\ 1 + 162x + 101x^2 + 168x^3 \end{bmatrix} \text{ and } v = t^T r + e_2 + \lceil 173/2 \rfloor m = 76 + 101x + 9x^2 + 26x^3.$$

The ciphertext is $c = (u, v)$.
DECRYPTION. To decrypt c, Bob uses her decryption key s to compute $v - s^T u = 85 + 2x + 72x^2 + 9x^3$ and rounds its coefficients to obtain $1 + x^2$, thereby recovering the plaintext $m = 1010$.

Does decryption work? Decryption succeeds if and only if $\text{Round}_q(v - s^T u) = m$. Now, substituting $u = A^T r + e_1$ and $v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m$ into $v - s^T u$ yields

$$\begin{aligned} v - s^T u &= (t^T r + e_2 + \lceil \frac{q}{2} \rfloor m) - s^T (A^T r + e_1) \\ &= (s^T A^T + e^T) r + e_2 + \lceil \frac{q}{2} \rfloor m - s^T (A^T r + e_1) \qquad \text{(after replacing } t^T \text{ by } s^T A^T + e^T) \\ &= e^T r + e_2 - s^T e_1 + \lceil \frac{q}{2} \rfloor m. \end{aligned}$$

Thus, $\text{Round}_q(v - s^T u) = m$ provided that each coefficient E_i of the *error polynomial* $E(x) = e^T r + e_2 - s^T e_1$ satisfies $-q/4 < E_i \bmod q < q/4$, i.e., $\|E_i\|_\infty < q/4$. Notice that e, e_1, e_2, r, and s all have small coefficients, so one can hope that $E(x)$ also has small coefficients (cf. Theorem 15.14). Indeed, since $\|e\|_\infty \le \eta_2$ and $\|r\|_\infty \le \eta_1$, we have $\|e^T r\|_\infty \le kn\eta_1\eta_2$. Similarly, $\|s^T e_1\|_\infty \le kn\eta_1\eta_2$. Hence, $\|E_i\|_\infty \le 2kn\eta_1\eta_2 + \eta_2$. Thus, a sufficient condition for decryption to succeed is $2kn\eta_1\eta_2 + \eta_2 < q/4$.

Example 15.23 (*decryption success probability*) The ML-KEM-768 parameters (Table 15.1) are $q = 3329$, $n = 256$, $k = 3$, $\eta_1 = 2$, $\eta_2 = 2$. For these parameters, $2kn\eta_1\eta_2 + \eta_2 = 6146 \not< \frac{q}{4}$, and hence decryption is not guaranteed to succeed. However, it can be shown that the probability that $\|E\|_\infty \ge \frac{q}{4}$ is less than 2^{-164}. Hence, the probability of decryption failure is negligibly small. Similarly, for the ML-KEM-1024 parameters (Table 15.1) $q = 3329$, $n = 256$, $k = 4$, $\eta_1 = 2$, $\eta_2 = 2$, the probability of decryption failure is less than 2^{-174}.

Security of Kyber-PKE. The security of Kyber-PKE is captured in the following theorem.

Theorem 15.24 The Kyber public-key encryption scheme is semantically secure against chosen-plaintext attack assuming that ss-D-MLWE$(k, \ell, n, q, \eta_1, \eta_2)$ is intractable.

Proof: The encryption operation can be written as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A^T \\ t^T \end{bmatrix} r + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor m \end{bmatrix}.$$

By the assumption of ss-D-MLWE intractability,

$$\begin{bmatrix} A^T \\ t^T \end{bmatrix}$$

is indistinguishable from random. Again, by the same assumption,

$$\begin{bmatrix} A^T \\ t^T \end{bmatrix} r + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} A^T r + e_1 \\ t^T r + e_2 \end{bmatrix}$$

is indistinguishable from random. Thus, from the adversary's perspective, v is the sum of the random polynomial $t^T r + e_2$ and the scaled message polynomial $\lceil \frac{q}{2} \rfloor m$. It is also clear that u is independent of m, and thus leaks nothing about m. Hence, the adversary learns nothing about m. □

Ciphertext compression. FIPS 203 specifies a method for compressing and decompressing Kyber-PKE ciphertexts. The approach involves discarding the "low order" bits of the coefficients of all polynomials in the ciphertext $c = (u, v)$. Since this compression is lossy, the compression parameters have been carefully selected to ensure that the errors introduced during decompression do not significantly affect the decryption error probability; see Fact 15.25.

Select a *compression parameter* $1 \leq d < \log_2 q$. For $X \in [0, q-1]$, define

$$\text{Compress}_q(X, d) = \lceil (2^d/q) \cdot X \rfloor \bmod 2^d. \tag{15.6}$$

For $Y \in [0, 2^d - 1]$, define

$$\text{Decompress}_q(Y, d) = \lceil (q/2^d) \cdot Y \rfloor \bmod q. \tag{15.7}$$

The decompression error is quantified next.

Fact 15.25 Let $d \in [1, \lfloor \log_2 q \rfloor]$, $X \in [0, q-1]$, $Y = \text{Compress}_q(X, d)$, and $X' = \text{Decompress}_q(Y, d))$. Then $\|X' - X\|_\infty \leq \lceil q/2^{d+1} \rfloor$.

The functions Compress and Decompress extend to polynomials in R_q and polynomial vectors in R_q^k by applying them to each coefficient of a polynomial.

Example 15.26 Let $q = 3329$ and $d = 10$. If $X \in [0, q-1]$, $Y = \text{Compress}_q(X, d)$, and $X' = \text{Decompress}_q(Y, d)$, then $\|X' - X\|_\infty \leq 2$. Also, if $U = 223 + 1438x + 3280x^2 + 798x^3 \in \mathbb{Z}_q[x]/(x^4 + 1)$, then $V = \text{Compress}_q(U, 10) = 69 + 442x + 1009x^2 + 245x^3$ and $U' = \text{Decompress}_q(V, 10) = 224 + 1437x + 3280x^2 + 796x^3$. We have $U - U' = -1 + x + 2x^3$.

FIPS 203 specifies two compression factors d_u and d_v (see Table 15.1). A Kyber-PKE ciphertext $c = (u, v)$ is compressed to $c' = (c_1, c_2)$ where $c_1 = \text{Compress}_q(u, d_u)$ and $c_2 = \text{Compress}_q(v, d_v)$. To decrypt $c' = (c_1, c_2)$, one first decompresses c_1 and c_2 to obtain $u' = \text{Decompress}_q(c_1, d_u)$ and $v' = \text{Decompress}_q(c_2, d_v)$, and then computes $m = \text{Round}_q(v' - s^T u')$.

15.3.5 The Kyber key encapsulation mechanism

Kyber-KEM is a quantum-safe key encapsulation mechanism standardized by NIST in FIPS 203. Its official name is Module-Lattice-based KEM (ML-KEM). As shown in Alg. 15.5, Kyber-KEM is derived by applying a variant of the Fujisaki-Okamoto (FO) transform to the Kyber public-key encryption scheme. The FO transform is a generic method for converting a public-key encryption scheme that is secure against chosen-plaintext attacks to one that is secure against chosen-ciphertext attacks.

The transform utilizes three hash function, G, H and J, which are derived from SHA3 and SHAKE256. Kyber-PKE is employed to encrypt a randomly chosen bit string m of length 256. During encapsulation, *derandomization* is applied, whereby m and the encapsulation key ek are hashed to produce a random seed R and the secret key K. The random polynomial vectors r, e_1 and the random polynomial e_2 required for Kyber-PKE are derived from R. This process transforms the randomized Kyber public-key encryption scheme into a deterministic one, meaning the ciphertext c depends only on the m and encapsulation key ek, without relying on any additional random bits generated during encryption.

In decapsulation, the recipient decrypts the Kyber-PKE ciphertext c to recover m', then hashes m' and ek to obtain R' and K'. She then re-encrypts m' using R' and compares the resulting ciphertext c' with the received c. If the encryption was performed correctly, then m' will equal m and consequently $R' = R$ and $K' = K$. This ensures that the re-encryption of m results in the ciphertext c, which is why a deterministic encryption scheme is necessary. However, if the encryption was not correctly executed, the ciphertext c' will differ from c with very high probability.

Therefore, if c' equals c, Bob accepts K as the shared key; otherwise, he generates a random key $\overline{K}$ obtained by hashing c and the secret z.

Kyber-KEM exhibits *plaintext awareness*, meaning that decapsulation will produce K (and not $\overline{K}$) only if the entity who performed the encapsulation knew K. This property is designed to ensure resistance to chosen-ciphertext attacks because the adversary cannot gain any useful information by submitting a ciphertext to Bob for decapsulation. Either the adversary already knew K (in which case nothing new is learned when Bob returns the same K), or the adversary does not know K, in which case Bob will almost certainly return a key $\overline{K}$ that is randomly selected and independent of K, preventing the adversary from learning anything useful.

Decapsulation fails when c does not equal to c', in which case the key $\overline{K}$ outputted is different from the encapsulated key K. This failure can occur even if Alice and Bob behave honestly, as there is is very small chance that a failure in the underlying Kyber-

PKE will cause m' to differ from m. However, for the domain parameters specified in FIPS 203, the probability of Kyber-PKE decryption failure is negligible, and consequently, the probability of a Kyber-KEM failure is also extremely low.

Algorithm 15.5: Kyber key encapsulation mechanism.

Domain parameters. Kyber-PKE parameters q, $n = 256$, k, η_1, η_2, and hash functions $G : \{0,1\}^* \rightarrow \{0,1\}^{2n}$, $H : \{0,1\}^* \rightarrow \{0,1\}^n$, $J : \{0,1\}^* \rightarrow \{0,1\}^n$.

Key generation. Bob does the following:

1 Use the Kyber-PKE key generation algorithm to select a Kyber-PKE encryption key (A, t) and decryption key s.
2 Select $z \in_R \{0,1\}^{256}$.
3 Bob's encapsulation (public) key is $ek = (A, t)$; her decapsulation (private) key is $dk = (s, ek, H(ek), z)$.

Encapsulation. To establish an n-bit shared secret key with Bob, Alice does:

4 Obtain an authentic copy of Bob's encapsulation key ek.
5 Select $m \in_R \{0,1\}^{256}$.
6 Compute $h = H(ek)$ and $(K, R) = G(m, h)$ where $K, R \in \{0,1\}^{256}$.
7 Use the Kyber-PKE encryption algorithm to encrypt m with ek, and using R to derive the random quantities needed for encryption; the resulting ciphertext is c.
8 Output the secret key K and ciphertext c.

Decapsulation. To recover the secret key K from c using $dk = (s, ek, H(ek), z)$, Bob does the following:

9 Using the Kyber-PKE decryption algorithm to decrypt c using decryption key s; the resulting plaintext is m'.
10 Compute $(K', R') = G(m', H(ek))$.
11 Compute $\overline{K} = J(z, c)$.
12 Use the Kyber-PKE enc. algorithm to encrypt m' with ek, and using R' to derive the random quantities needed for encryption; the resulting ciphertext is c'.
13 If $c \neq c'$ then return($\overline{K}$).
14 Return(K').

Security. Kyber-KEM has been proven to be indistinguishable against chosen-ciphertext attacks assuming that D-MLWE is intractable, and that G, H, and J behave as random functions. Kyber-KEM has also been proven indistinguishable against chosen-ciphertext attacks by a quantum adversary, who is also able to make quantum queries (in superposition) to G, H and J.

15.4 Dilithium Signature Scheme

Dilithium is a quantum-safe signature scheme standardized by NIST in FIPS 204, where it is officially named Module-Lattice-based Digital Signature Algorithm (ML-DSA). It is

designed to provide strong security guarantees against attacks from both classical and quantum computers. In §15.4.2, we'll introduce a "toy version" of Dilithium to illustrate its core principles. We'll then identify some key weaknesses in this simplified scheme and, in §15.4.3, refine it by addressing these issues—bringing us closer to the standardized Dilithium scheme.

The security of Dilithium relies on the hardness of the Decisional Module Learning With Errors Problem (D-MLWE) and the module variant (MSIS) of the Short Integer Solutions problem, which is described next.

15.4.1 Short integer solutions problem

The Short Integer Solutions (SIS) problem was introduced by Miklós Ajtai in 1996.

Notation. q is a prime, $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$ is the set of integers modulo q, $\mathbb{Z}_q^m$ is the set of length-m (column) vectors whose components are in $\mathbb{Z}_q$, and $\mathbb{Z}_q^{n\times m}$ is the set of $n \times m$ matrices whose entries are in $\mathbb{Z}_q$.

Definition 15.27 The *Short Integer Solutions* problem SIS(n, m, q, B) is the following. Given $A \in_R \mathbb{Z}_q^{n\times m}$, find $z \in \mathbb{Z}^m$ such that $Az = 0 \pmod q$, where $z \neq 0$ and $z \in [-B, B]^m$; see Figure 15.12. Here, $B \ll q/2$.

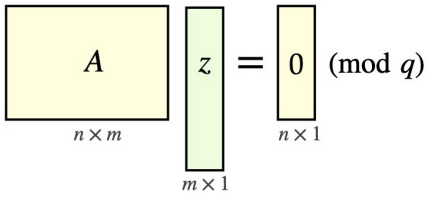

Figure 15.12: SIS problem.

SIS seeks to find a small nonzero solution in the null space of $A \pmod q$. While Gaussian elimination can efficiently determine a basis for the null space, the challenge is to find a nonzero vector in that null space all of whose coordinates are small, i.e., within the interval $[-B, B]$.

When $n \geq m$, we expect that A has rank m, meaning that $z = 0$ is the only solution to $Az = 0 \pmod q$, and thus no SIS solution exists. Therefore, we henceforth assume that $n < m$.

If $(B+1)^m > q^n$, the pigeonhole principle guarantees that there must exist distinct $z_1, z_2 \in [-B/2, B/2]^m$ such that $Az_1 = Az_2 \pmod q$. In this case, the difference $z = z_1 - z_2$ is an SIS solution. Henceforth, we'll assume that $(B+1)^m > q^n$, or equivalently, $m > (n \log q)/\log(B+1)$, ensuring that an SIS solution exists. It is important to note, however, that an SIS solution is not unique. Indeed, if z is one SIS solution, then so is $-z \bmod q$.

Example 15.28 (*SIS instance*) Let $n = 3$, $m = 5$, $q = 13$, and $B = 3$. Consider the SIS instance

$$A = \begin{bmatrix} 1 & 2 & 9 & 0 & 4 \\ 2 & 11 & 3 & 10 & 12 \\ 10 & 8 & 10 & 5 & 1 \end{bmatrix} \in \mathbb{Z}_{13}^{3\times 5}.$$

We need to find nonzero $z = (z_1, z_2, z_3, z_4, z_5) \in [-3, 3]^5$ such that $Az = 0 \pmod{13}$.

Gaussian elimination (mod 13) on A yields the reduced matrix

$$A' = \begin{bmatrix} 1 & 0 & 0 & 3 & 4 \\ 0 & 1 & 0 & 3 & 5 \\ 0 & 0 & 1 & 12 & 9 \end{bmatrix}.$$

The rows of A' each provide one equation, and thus the complete solution to $Az = 0 \pmod{13}$ is $z_1 = 10z_4 + 9z_5$, $z_2 = 10z_4 + 8z_5$, $z_3 = z_4 + 4z_5$, $z_4 \in \mathbb{Z}_{13}$, $z_5 \in \mathbb{Z}_{13}$. Among the $13^2 = 169$ solutions $z \in \mathbb{Z}_{13}^5$ are 10 SIS solutions (easily found by trial and error): $z = \pm(1, -2, -1, 0, 3)$, $z = \pm(3, 3, -1, -1, 0)$, $z = \pm(1, 2, -3, 1, -1)$, $z = \pm(1, 3, 3, -2, -2)$ and $z = \pm(2, 1, 2, -2, 1)$.

Definition 15.29 The *Inhomogeneous Short Integer Solutions* problem ISIS(n, m, q, B) is the following. Given $A \in_R \mathbb{Z}_q^{n\times m}$ and $b \in_R \mathbb{Z}_q^n$, find $z \in \mathbb{Z}^m$ such that $Az = b \pmod{q}$ and $z \in [-B, B]^m$. Here, $B \ll q/2$.

We'll assume that $n < m$, whereby the matrix A is expected to have full rank. We'll also assume that $(2B + 1)^m \gg q^n$, because then an ISIS solution is likely to exist.

Theorem 15.30 SIS and ISIS are computationally equivalent.

Proof: (ISIS $\leq$ SIS) Let (A, b) be an ISIS(n, m, q, B) instance. Select $j \in_R [1, m + 1]$ and $c \in_R [-B, B]$ with $c \neq 0$. Let A' be the $n \times (m + 1)$ matrix obtained by inserting $-c^{-1}b \bmod q$ as a new jth column in A. Now, use an SIS solver to find an SIS solution $z' \in [-B, B]^{m+1}$ to $A'z' = 0 \pmod{q}$. If indeed the jth entry in z' is c, then $Az = b \pmod{q}$, where $z \in [-B, B]^m$ is obtained from z' by deleting its jth entry. Thus, z is an ISIS solution that we have efficiently found with significant probability approximately $1/(2Bm)$.

(SIS $\leq$ ISIS) Let A be an SIS(n, m, q, B) instance. Write $A = [A' | -b']$ where $A' \in \mathbb{Z}_q^{n\times(m-1)}$ and $b' \in \mathbb{Z}_q^n$. Use an ISIS solver to find an ISIS solution z' to the ISIS instance (A', b'). We then have $A'z' = b' \pmod{q}$ and $z' \in [-B, B]^{m-1}$. Then $z = (z', 1) \in \mathbb{Z}^m$ satisfies $Az = 0 \pmod{q}$, $z \neq 0$, and $z \in [-B, B]^m$. Thus, z is an SIS solution that we have efficiently found. □

Definition 15.31 The *normal-form ISIS* problem nf-ISIS(n, m, q, B) is the following. Given $A \in_R \mathbb{Z}_q^{n\times m}$ and $b \in_R \mathbb{Z}_q^n$, find $z \in \mathbb{Z}^{m+n}$ such that $[A \,|\, I_n]z = b \pmod{q}$ and $z \in [-B, B]^{m+n}$. Here, I_n is the $n \times n$ identity matrix and $B \ll q/2$.

Exercise 15.30 shows that nf-ISIS(n, m, q, B) and ISIS$(n, m+n, q, B)$ are computationally equivalent.

Hardness of SIS. In his 1996 paper, Mikló Ajtai proved that if approx-SIVP is hard in the worst case (against classical attacks), then SIS is hard on average. Similar to Regev's worst-case to average-case reduction for LWE, Ajtai's reduction is very inefficient and only provides an asymptotic hardness guarantee for SIS.

The best known approach for solving SIS in practice is to reframe the problem as an instance of approx-SVP, and then apply the fastest available approx-SVP solvers, all of which have fully-exponential running times. The performance of these approx-SVP solvers is used to determine appropriate parameters for SIS-based cryptosystems in real-world applications.

The Module-SIS problem. MSIS is a variant of SIS where integers modulo q are replaced by polynomials in the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Given a set of randomly selected elements in the module R_q^k, MSIS involves finding a linear combination of the polynomial vectors that equals 0, with the scalar multipliers of the linear combination being small polynomials. The matrix representation of MSIS is as follows.

Definition 15.32 The *Module Short Integer Solutions* problem MSIS(k, ℓ, n, q, B) is the following. Given $A \in_R R_q^{k \times \ell}$, find $z \in R_q^\ell$ such that $Az = 0$, where $z \neq 0$ and $\|z\|_\infty \leq B$. Here, $k < \ell$ and $B \ll q/2$.

Note that an MSIS solution is not unique. Indeed, if $z = (z_1, z_2, \ldots, z_\ell)$ is one MSIS solution, then so is $(xz_1, xz_2, \ldots, xz_\ell)$. Thus, an MSIS solution yields as many as $2n$ other distinct solutions.

Example 15.33 (*MSIS instance*) Let $k = 2$, $\ell = 3$, $n = 4$, $q = 71$ (so $R_q = \mathbb{Z}_{71}[x]/(x^4+1)$), and $B = 10$. Consider the MSIS instance

$$A = \begin{bmatrix} 16 + 58x^2 + 30x^3 & 22 + 40x + 4x^2 + 57x^3 & 53 + 8x + 21x^2 + 38x^3 \\ 61 + 32x + 3x^2 + 45x^3 & 23 + 5x + 58x^2 + 40x^3 & 49 + 40x + 29x^2 + 54x^3 \end{bmatrix} \in R_q^{2\times 3}.$$

Given A, the task is to find nonzero $z = (z_1, z_2, z_3) \in R_q^3$ such that $Az = 0$ and $\|z\|_\infty \leq 10$.

Recall that polynomial multiplication in R_q can be expressed as a matrix-vector product. Consequently, an equivalent formulation of this MSIS instance is to solve $\overline{A}\overline{z} = 0 \pmod{71}$ for nonzero $\overline{z} \in [-10, 10]^{12}$, where

$$\overline{A} = \left[\begin{array}{cccc|cccc|cccc} 16 & 41 & 13 & 0 & 22 & 14 & 67 & 31 & 53 & 33 & 50 & 63 \\ 0 & 16 & 41 & 13 & 40 & 22 & 14 & 67 & 8 & 53 & 33 & 50 \\ 58 & 0 & 16 & 41 & 4 & 40 & 22 & 14 & 21 & 8 & 53 & 33 \\ 30 & 58 & 0 & 16 & 57 & 4 & 40 & 22 & 38 & 21 & 8 & 53 \\ \hline 61 & 26 & 68 & 39 & 23 & 31 & 13 & 66 & 49 & 17 & 42 & 31 \\ 32 & 61 & 26 & 68 & 5 & 23 & 31 & 13 & 40 & 49 & 17 & 42 \\ 3 & 32 & 61 & 26 & 58 & 5 & 23 & 31 & 29 & 40 & 49 & 17 \\ 45 & 3 & 32 & 61 & 40 & 58 & 5 & 23 & 54 & 29 & 40 & 49 \end{array}\right]_{8\times 12}.$$

The 4×4 blocks in $\overline{A}$ are the anti-circulant matrices corresponding to the polynomials in A.

Gaussian elimination (mod 71) on $\overline{A}$ yields the following rank-8 matrix in reduced form:

$$\tilde{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 37 & 4 & 48 & 8 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 63 & 37 & 4 & 48 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 23 & 63 & 37 & 4 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 67 & 23 & 63 & 37 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 52 & 19 & 47 & 54 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 17 & 52 & 19 & 47 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 24 & 17 & 52 & 19 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 52 & 24 & 17 & 52 \end{bmatrix}.$$

Since the null space of $\overline{A}$ has dimension 4, the total number of solutions $\overline{z}$ to $\overline{A}\overline{z} = 0$ (mod 71) is $71^4 = 25,411,681$. Checking all these solutions $\overline{z}$ yields 16 nonzero solutions all whose coordinates are in $[-10, 10]$. The two MSIS solutions, up to multiplication by $\pm 1, \pm x, \pm x^2, \pm x^3$, are $\overline{z} = (0, -5, 9, 1, -10, 4, -10, 10, -9, 5, 9, -6)$ and $\overline{z} = (3, -10, -7, -4, 5, 0, 1, 5, 4, -9, 6, -6)$. The polynomial form of the first solution is $z = (-5x + 9x^2 + x^3,\ -10 + 4x - 10x^2 + 10x^3,\ -9 + 5x + 9x^2 - 6x^3)$.

Hardness of MSIS. No attacks are currently known on MSIS that are faster than the most efficient attacks on SIS. In other words, no attacks on MSIS have been discovered that take advantage of the special structure of the matrix A, which consists of blocks of anti-circulant matrices.

15.4.2 Dilithium (toy version)

The Schnorr signature scheme (§9.6.2), which served as the foundation for EdDSA, also inspired the design of Dilithium. We'll begin by presenting an initial attempt at describing an analogue to Schnorr, based on the hardness of the MLWE and MSIS problems. The initial attempt will then be refined to create a "toy version" of Dilithium, and we'll identify some weaknesses with this toy version.

For the initial attempt, we use parameters q, n, k, ℓ, η, γ_1 and τ. Alice randomly selects a matrix $A \in_R R_q^{k \times \ell}$ with polynomial entries, along with two vectors $s_1 \in_R S_\eta^\ell$ and $s_2 \in_R S_\eta^k$, where the polynomial entries of s_1 and s_2 have small coefficients. (Recall that S_η denotes the set of polynomials $f \in R_q$ with $\|f\|_\infty \leq \eta$.) She then computes $t = As_1 + s_2 \in R_q^k$. Alice's verification (public) key is (A, t), and her signing (private) key is (s_1, s_2). Observe that deriving the private key from the public key is an instance of ss-MLWE, the short-secret variant of MLWE (Definition 15.20).

To sign a message $M \in \{0,1\}^*$, Alice selects secret $y \in_R S_{\gamma_1}^\ell$ and computes the *commitment* $w = Ay \in R_q^k$. Here, γ_1 is significantly smaller than $q/2$, meaning that computing y from w is an instance of the inhomogeneous variant of MSIS (Exercise 15.34). Alice then computes the *challenge* $c = H(M\|w)$, where $H : \{0,1\}^* \longrightarrow B_\tau$ is a hash function whose bitstring output is interpreted to represent an element in B_τ, the set of polynomials in

S_1 exactly τ of whose mods q coefficients are ± 1. Finally, Alice computes the *response* $z = y + cs_1 \in R_q^\ell$. Alice's signature on M is $\sigma = (c, z)$, which differs from Schnorr's signatures as outlined in §9.6.2, where the signature consists of the commitment (versus a challenge here) and response.

The polynomial vector cs_1 is intended to conceal y, even though c and z are public. Let $\beta = \tau\eta$. Since $c \in B_\tau$ and $s_1 \in S_\eta^\ell$, it follows $\|cs_1\|_\infty \leq \beta$. The integer parameters γ_1, η and τ are chosen so that $\gamma_1 + \beta < q/2$. Thus, $\|z\|_\infty = \|y + cs_1\|_\infty \leq \gamma_1 + \beta < q/2$.

To verify Alice's signature $\sigma = (c, z)$ on M, an entity Bob, who possesses Alice's public key (A, t), must first compute the commitment w and then check if $c = H(M\|w)$. Since $z = y + cs_1$, we have $Az = Ay + c(As_1) = w + c(t - s_2)$, which simplifies to

$$Az - ct = w - cs_2. \tag{15.8}$$

Therefore, Bob can compute $w - cs_2$, but not w itself. However, note that the coefficients of cs_2 are small, specifically $\|cs_2\|_\infty \leq \beta$. To address the issue of Bob being unable to to compute w, we'll introduce the notions of "HighBits" and "LowBits" applied to the mods q representation of an integer. These notions, which will be explained in §15.4.3, are analogous to the "most significant bits" and "least significant bits" in the standard binary representation of an integer. Using this approach, one then defines HighBits(g) and LowBits(g) for a polynomial vector g by applying the HighBits and LowBits operations to all coefficients of all polynomials in g.

Thus, the signing procedure is modified as follows to yield our toy version. Alice repeatedly selects $y \in_R S_{\gamma_1}^\ell$ until the LowBits of all coefficients of the polynomials in $w - cs_2$ are "sufficiently small" (this notion will be made precise later), to ensure that adding cs_2 to $w - cs_2$ doesn't alter any HighBits, i.e., HighBits$(w - cs_2)$ = HighBits(w). She then computes the commitment w_1 = HighBits(w), the challenge $c = H(M\|w_1)$, and the response $z = y + cs_1$. As before, her signature on M is $\sigma = (c, z)$. The verifier, Bob, can now compute the commitment w_1 since

$$w_1 = \text{HighBits}(w) = \text{HighBits}(w - cs_2) = \text{HighBits}(Az - ct),$$

and verify that the challenge was correctly computed from M and w_1. The signature scheme is presented in Alg. 15.6. Table 15.2 gives the FIPS 204 parameter sets.

In Alg. 15.6, y is sampled from $\tilde{S}_{\gamma_1}^\ell$, the set of polynomial vectors in R_q^ℓ all of whose coefficients are in $(-\gamma_1, \gamma_1]$. Notice that $-\gamma_1$ is excluded from this interval. Therefore, each coefficient has $2 * \gamma_1$ possible values, which is a power of two for $\gamma_1 = 2^{17}$ and $\gamma_1 = 2^{19}$ as in Table 15.2. Thus, the coefficients for candidate polynomials in y can be selected by randomly generating binary strings of length 17 or 19.

Next, we examine five issues with the toy version of Dilithium and modifications that address them.

Issue #1: Large public keys. For the ML-DSA-87 parameters, the matrix $A \in R_q^{k \times \ell}$ has size 41,216 bytes, and the vector $t \in R_q^k$ has size 5,888 bytes, so the public key (A, t) has size 47,104 bytes. To reduce this, A is generated from a randomly selected 256-bit

Algorithm 15.6: Dilithium signature scheme (toy version)

Domain parameters. q, n, k, ℓ, η, γ_1, τ (see Table 15.2).

Key generation. Alice does the following:

1 Select $A \in_R R_q^{k \times \ell}$, $s_1 \in_R S_\eta^\ell$, and $s_2 \in_R S_\eta^k$.
2 Compute $t = As_1 + s_2$.
3 Alice's verification (public) key is (A, t); her signing (private) key is (s_1, s_2).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

4 Found $\leftarrow$ false.
5 **while** Found = false **do**
6 Select $y \in_R \tilde{S}_{\gamma_1}^\ell$.
7 Compute $w = Ay$ and $w_1 = \text{HighBits}(w)$.
8 Compute $c = H(M \| w_1)$.
9 Compute $z = y + cs_1$.
10 **if** LowBits$(w - cs_2)$ are "sufficiently small" **then**
11 Found $\leftarrow$ true.
12 Return$(\sigma \leftarrow (c, z))$.

Signature verification. To verify Alice's signature $\sigma = (c, z)$ on M, Bob does:

13 Obtain an authentic copy of Alice's verification key (A, t).
14 Compute $w_1' = \text{HighBits}(Az - ct)$.
15 Accept if and only if $c = H(M \| w_1')$.

Table 15.2: FIPS 204 parameter sets for Dilithium (ML-DSA).

Parameter	Description	ML-DSA-44	ML-DSA-65	ML-DSA-87
q	prime modulus	$2^{23} - 2^{13} + 1$	$2^{23} - 2^{13} + 1$	$2^{23} - 2^{13} + 1$
n	$R_q = \mathbb{Z}_q[x]/(x^n + 1)$	256	256	256
(k, ℓ)	dimensions of A	(4,4)	(6,5)	(8,7)
η	coefficient bound for s_1, s_2	2	4	2
γ_1	coefficient bound for y	2^{17}	2^{19}	2^{19}
τ	number of 1's in c	39	49	60
β	$\beta = \tau\eta$	78	196	120
γ_2	HighBits/LowBits parameter	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
α	$\alpha = 2\gamma_2$	$(q-1)/44$	$(q-1)/16$	$(q-1)/16$
λ	collision strength of $\tilde{c}$	128	192	256

seed ρ, and its entries are derived from ρ using a publicly-known pseudorandom bit generator called ExpandA. The updated public key, (ρ, t), has size 5,920 bytes.

FIPS 204 also specifies a method for compressing t, reducing its size from 5,888 bytes to 2,560 bytes, further reducing the public key size to 2,592 bytes.

Issue #2: Large private keys. For the ML-DSA-87 parameters, the secret key $(s_1, s_2) \in S_\eta^\ell \times S_\eta^k$ has a size of 1,440 bytes. To reduce this, s_1 and s_2 are generated from a randomly selected 512-bit secret seed ρ' using a pseudorandom bit generator called ExpandS.

Issue #3: Repeated hashing of M. If the message M is very large, then the hash operation $H(M\|w_1)$ in step 2.3 will be the most time consuming operation in signature generation. Moreover, step 2.3 might be executed several times. To reduce the time cost, a message representative μ is first computed before the while loop by hashing M, and then μ is used in place of M in step 2.3.

Issue #4: Random bits required for signing. A large number of random bits are needed to generate $y \in \tilde{S}_{\gamma_1}^\ell$ in each iteration of the signature generation process. To eliminate the need for random bits, y is generated using a secret seed ρ'' and a counter κ via a pseudorandom bit generator called ExpandMask. The seed ρ'' is derived by hashing a secret key K and the message hash μ. This approach makes the signature process *deterministic*, meaning a signature is solely dependent on the message and the signing key. Additionally, FIPS 204 allows the signer to supplement the input to ExpandMask with a 256-bit randomly generated string. If this option is chosen, then the signing process becomes *randomized*.

Issue #5: A signature (c, z) may reveal information about s_1. Since $y \in \tilde{S}_{\gamma_1}^\ell$, we have $-\gamma_1 < \text{coeff}(y) \leq \gamma_1$ where coeff(y) denotes the mods q representation of an arbitrary coefficient of an arbitrary polynomial in y. Also, since $c \in B_\tau$ and $s_1 \in S_\eta^\ell$, we have $-\beta \leq \text{coeff}(cs_1) \leq \beta$ where $\beta = \tau\eta$. Thus, since $z = y + cs_1$ and $\gamma_1 + \beta < q/2$, we have $-\gamma_1 - \beta < \text{coeff}(z) \leq \gamma_1 + \beta$.

Now, if a coefficient of a polynomial in z equals $\gamma_1 + \beta$, the corresponding coefficient of cs_1 must be β. This leaks *some* information about the coefficients of the private key s_1.

Similarly, if a coefficient of a polynomial in z is $\gamma_1 + \beta - 1$, the corresponding coefficient of cs_1 must be either $\beta - 1$ or β, again leaking *some* information about s_1.

More generally (as shown in Figure 15.13), if a coefficient of a polynomial in z is $\gamma_1 + a$ for some $-\beta + 1 \leq a \leq \beta$ (or the coefficient is $-\gamma_1 + a$ for some $-\beta < a \leq \beta$), the corresponding coefficient of cs_1 must fall within the interval $[a, \beta]$ (resp. in $[-\beta, a-1]$).

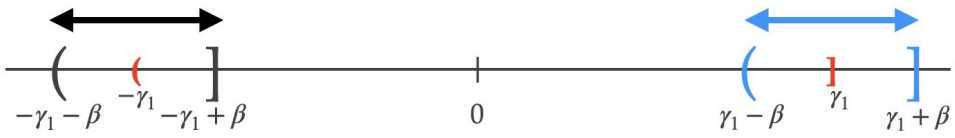

Figure 15.13: Rejection sampling in Dilithium.

To prevent this leakage of information about s_1, *rejection sampling* is employed. The signer repeatedly selects $y \in_R \tilde{S}^{\ell}_{\gamma_1}$ and computes $z = y + cs_1$ until $-\gamma_1 + \beta < \text{coeff}(z) \leq \gamma_1 - \beta$; FIPS 204 imposes the slightly stricter constraint that $\|z\|_\infty < \gamma_1 - \beta$. Rejection sampling aims to ensure that z does not reveal any information about cs_1, and thus does not leak information about s_1.

15.4.3 Dilithium (without t compression)

This section presents a slightly simplified version of the standardized Dilithium signature scheme (also known as ML-DSA), taking into account the discussion of the previous section. We'll begin by defining the notions of HighBits and LowBits of an integer modulo q.

HighBits and LowBits. Let α be an even divisor of $q-1$, and $m = (q-1)/\alpha$. For $r \in [0, q-1]$, define $r_0 = r \bmod \alpha$ and $r_1 = (r - r_0)/\alpha$; note that $r = r_1\alpha + r_0$ with $0 \leq r_1 \leq m$ and $-\alpha/2 < r_0 \leq \alpha/2$. Then $\text{HighBits}(r, \alpha) = r_1$ and $\text{LowBits}(r, \alpha) = r_0$. Figure 15.14 depicts the HighBits and LowBits of all $r \in [0, q-1]$ for the case $q = 21$ and $\alpha = 4$. For a polynomial vector $w \in R_q^{\ell}$, $\text{HighBits}(w, \alpha)$ and $\text{LowBits}(w, \alpha)$ are obtained by applying HighBits and LowBits to each coefficient of each polynomial in w.

Alg. 15.7 incorporates the solutions to the five issues with the toy version of Dilithium, as discussed in §15.4.2. Further details of the protocol, which we omit from Alg. 15.7, can be found in FIPS 204. Specifically, the pseudorandom bit generator ExpandA is implemented using SHAKE128, while the pseudorandom bit generators ExpandS and ExpandMask are implemented using SHAKE256. The variable-output length hash function H, which takes an input $m \in \{0,1\}^*$ and a length parameter $d \geq 1$ to output $H(m, d) \in \{0,1\}^d$, is instantiated using either SHAKE128 or SHAKE256. Additionally, the function SampleInBall maps bitstrings in $\{0,1\}^{2\lambda}$ to polynomials in B_τ.

Number of iterations in signature generation. The Dilithium parameter sets in FIPS 204 were carefully selected so that the expected number of iterations of the main loop (step 5) of signature generation remains small. For the full version of Dilithium, the

Algorithm 15.7: Dilithium signature scheme (without t compression)

Domain parameters. $q, n, k, \ell, \eta, \gamma_1, \gamma_2, \tau, \beta, \lambda$ (see Table 15.2).

Key generation. Alice does the following:

1 Select $\xi \in_R \{0,1\}^{256}$.
2 Compute $(\rho, \rho', K) = H(\xi, 1024)$, where $\rho \in \{0,1\}^{256}$, $\rho' \in \{0,1\}^{512}$, $K \in \{0,1\}^{256}$.
3 Compute $A = \text{ExpandA}(\rho)$.
4 Compute $(s_1, s_2) = \text{ExpandS}(\rho')$.
5 Compute $t = As_1 + s_2$.
6 Compute $tr = H(\rho \| t,\ 2\lambda)$.
7 Alice's verification (public) key is (ρ, t); her signing (private) key is (ρ, K, tr, s_1, s_2).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

8 Compute $A = \text{ExpandA}(\rho)$.
9 Compute $\mu = H(tr \| M,\ 512)$.
10 Compute $\rho'' = H(K \| \text{rnd} \| \mu,\ 512)$ where either $\text{rnd} = 0^{256}$ or $\text{rnd} \in_R \{0,1\}^{256}$.
11 Set $\kappa = 0$ and Found = false.
12 **while** Found = false **do**
13 Compute $y = \text{ExpandMask}(\rho'', \kappa)$.
14 Compute $w = Ay$ and $w_1 = \text{HighBits}(w, 2\gamma_2)$.
15 Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$.
16 Compute $c = \text{SampleInBall}(\tilde{c})$ and $z = y + cs_1$.
17 Compute $r_0 = \text{LowBits}(w - cs_2, 2\gamma_2)$.
18 **if** $\|z\|_\infty < \gamma_1 - \beta$ and $\|r_0\|_\infty < \gamma_2 - \beta$ **then**
19 Found $\leftarrow$ true.
20 $\kappa \leftarrow \kappa + \ell$ (increment the counter κ).
21 Return$(\sigma \leftarrow (\tilde{c}, z))$.

Signature verification. To verify Alice's signature $\sigma = (\tilde{c}, z)$ on M, Bob does:

22 Obtain an authentic copy of Alice's verification key $PK = (\rho, t)$.
23 Check that $\|z\|_\infty < \gamma_1 - \beta$; if not then reject the signature.
24 Compute $A = \text{ExpandA}(\rho)$.
25 Compute $tr = H(\rho \| t, 512)$ and $\mu = H(tr \| M, 512)$.
26 Compute $c = \text{SampleInBall}(\tilde{c}_1)$.
27 Compute $w_1' = \text{HighBits}(Az - ct, 2\gamma_2)$.
28 Check that $\tilde{c} = H(\mu \| w_1', 2\lambda)$; if not then reject the signature.
29 Accept the signature.

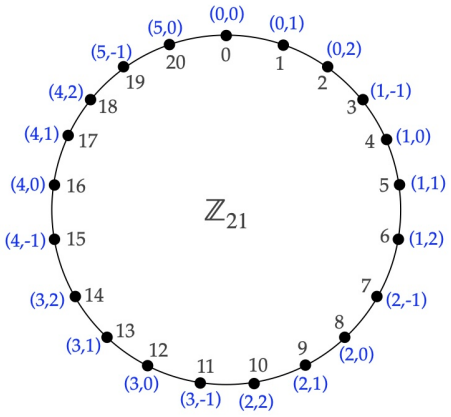

Figure 15.14: (HighBits$(r, 4)$, LowBits$(r, 4)$) for $q = 21$ and $0 \le r \le 20$.

expected number of iterations is 4.25, 5.1 and 3.85 for the ML-DSA-44, ML-DSA-65, and ML-DSA-87 parameter sets, respectively.

Example 15.34 (*toy example of Dilithium signature scheme per Alg. 15.7*)
Domain parameters. $q = 59393$, $n = 5$, $k = 4$, $\ell = 3$, $\eta = 20$, $\gamma_1 = 2^{12} = 4096$, $\gamma_2 = (q-1)/32 = 1856$, $\tau = 4$, $\beta = 80$.
Key generation. Alice generates the following matrix $A \in R_q^{4\times 3}$:

$$\begin{bmatrix} 13443+15256x+19876x^2+59185x^3+28675x^4 & 36145+57667x+34116x^2+42016x^3+50380x^4 \\ & 28454+4544x+43159x^2+24596x^3+21893x^4 \\ 16032+28251x+2973x^2+15033x^3+28501x^4 & 13622+12022x+36427x^2+20264x^3+32304x^4 \\ & 33688+22466x+31555x^2+51570x^3+6890x^4 \\ 50932+22198x+4761x^2+23629x^3+28659x^4 & 24428+19490x+48399x^2+15890x^3+9598x^4 \\ & 11653+32589x+49653x^2+10612x^3+35396x^4 \\ 33551+13045x+58486x^2+1793x^3+36048x^4 & 991+52579x+46671x^2+31699x^3+54664x^4 \\ & 31133+25165x+48960x^2+55717x^3+50305x^4 \end{bmatrix}.$$

Alice then selects

$$s_1 = \begin{bmatrix} 8+14x-12x^2+20x^3+20x^4 \\ -5-5x+20x^2+2x^3-15x^4 \\ 1-6x-14x^2-15x^3+7x^4 \end{bmatrix} \in S_{20}^3, \quad s_2 = \begin{bmatrix} 12+20x+20x^2-17x^3+9x^4 \\ 4x+20x^2+5x^3+6x^4 \\ -1+15x-x^2+x^3-2x^4 \\ -1-18x+4x^2+10x^3+10x^4 \end{bmatrix} \in S_{20}^4,$$

and computes

$$t = As_1 + s_2 = \begin{bmatrix} 29545+13137x+31004x^2+36362x^3+53538x^4 \\ 15656+39462x+40541x^2+46135x^3+15684x^4 \\ 27717+31374x+39319x^2+4977x^3+41316x^4 \\ 58640+52868x+38983x^2+45544x^3+6501x^4 \end{bmatrix} \in R_q^4.$$

Here, the coefficients of polynomials in s_1 and s_2 are expressed in their mods q representation. Alice's verification key is (A, t); her signing key is (s_1, s_2).
SIGNATURE GENERATION. To sign a message M, Alice generates

$$y \bmod q = \begin{bmatrix} -514 - 2635x + 4019x^2 - 3328x^3 - 1600x^4 \\ -1032 + 3846x - 2272x^2 - 408x^3 - 2284x^4 \\ 3307 + 1789x + 2836x^2 + 3677x^3 - 74x^4 \end{bmatrix} \in \tilde{S}^3_{\gamma_1}$$

and computes

$$w = Ay = \begin{bmatrix} 34778 + 5179x + 27315x^2 + 8882x^3 + 46288x^4 \\ 28548 + 19871x + 20469x^2 + 31744x^3 + 18160x^4 \\ 55876 + 15135x + 5029x^2 + 22499x^3 + 51434x^4 \\ 23351 + 1108x + 47241x^2 + 44892x^3 + 45631x^4 \end{bmatrix}$$

and

$$w_1 = \text{HighBits}(w, 2\gamma_2) = \begin{bmatrix} 9 + x + 7x^2 + 2x^3 + 12x^4 \\ 8 + 5x + 6x^2 + 9x^3 + 5x^4 \\ 15 + 4x + x^2 + 6x^3 + 14x^4 \\ 6 + 13x^2 + 12x^3 + 12x^4 \end{bmatrix}.$$

Alice computes $\tilde{c} = H(\mu \| w_1)$ and $c = \text{SampleInBall}(\tilde{c})$ (details omitted), obtaining $c \bmod q = x^3 + x^2 - x - 1 \in B_4$, and then computes

$$z = y + cs_1 = \begin{bmatrix} 58883 + 56696x + 4005x^2 + 56079x^3 + 57755x^4 \\ 58329 + 3869x + 57116x^2 + 58953x^3 + 57137x^4 \\ 3342 + 1802x + 2850x^2 + 3701x^3 + 59307x^4 \end{bmatrix}.$$

Note that $\|z\|_\infty < \gamma_1 - \beta = 4016$. Alice also computes

$$w - cs_2 = \begin{bmatrix} 34784 + 5203x + 27352x^2 + 8853x^3 + 46240x^4 \\ 28567 + 19886x + 20499x^2 + 31765x^3 + 18147x^4 \\ 55877 + 15148x + 5042x^2 + 22485x^3 + 51419x^4 \\ 23354 + 1109x + 47238x^2 + 44925x^3 + 45665x^4 \end{bmatrix}$$

and

$$r_0 = \text{LowBits}(w - cs_2, 2\gamma_2) = \begin{bmatrix} 1376 + 1491x + 1368x^2 + 1429x^3 + 1696x^4 \\ -1129 + 1326x - 1773x^2 - 1643x^3 - 413x^4 \\ 197 + 300x + 1330x^2 + 213x^3 - 549x^4 \\ 1082 + 1109x - 1018x^2 + 381x^3 + 1121x^4 \end{bmatrix}.$$

Note that $\|r_0\|_\infty < \gamma_2 - \beta = 1776$. Alice's signature on M is $\sigma = (\tilde{c}, z)$.
SIGNATURE VERIFICATION. To verify Alice's signature $\sigma = (\tilde{c}, z)$ on M, Bob computes $c = \text{SampleInBall}(\tilde{c})$ obtaining $c \bmod q = x^3 + x^2 - x - 1 \in B_4$,

$$Az - ct = \begin{bmatrix} 34784 + 5203x + 27352x^2 + 8853x^3 + 46240x^4 \\ 28567 + 19886x + 24099x^2 + 31765x^3 + 18147x^4 \\ 55877 + 15148x + 5042x^2 + 22485x^3 + 51419x^4 \\ 23354 + 1109x + 47238x^2 + 44925x^3 + 45665x^4 \end{bmatrix}$$

and

$$w_1' = \text{HighBits}(Az - ct, 2\gamma_2) = \begin{bmatrix} 9 + x + 7x^2 + 2x^3 + 12x^4 \\ 8 + 5x + 6x^2 + 9x^3 + 5x^4 \\ 15 + 4x + x^2 + 6x^3 + 14x^4 \\ 6 + 13x^2 + 12x^3 + 12x^4 \end{bmatrix}.$$

Since $w_1' = w_1$, we have $H(\mu \| w_1', 2\lambda) = H(\mu \| w_1, 2\lambda)$ so Bob accepts the signature.

Signature verification works. Recall (equation 15.8) that $Az - ct = w - cs_2$. Since $\|\text{LowBits}(w - cs_2,\ 2\gamma_2)\|_\infty < \gamma_2 - \beta$ and $\|cs_2\|_\infty \le \beta$, we have

$$w_1' = \text{HighBits}(Az - ct,\ 2\gamma_2) = \text{HighBits}(w - cs_2,\ 2\gamma_2) = \text{HighBits}(w,\ 2\gamma_2) = w_1.$$

Thus, $H(\mu \| w_1',\ 2\lambda) = H(\mu \| w_1,\ 2\lambda)$ and the signature will be accepted.

Security of Dilithium (Alg. 15.7). If the short-secret D-MLWE problem (Definition 15.21) is intractable, an adversary cannot derive any information about the secret key component s_1 (or s_2) from the public key (ρ, t). Evaluating the difficulty of signature forgery is more complex. To simplify the analysis, we'll make a few simplifications to the signature scheme and provide an informal argument that a forger with access to the public key (ρ, t) cannot produce a valid signed message.

The forger's task is as follows: given (ρ, t), find $(M, \tilde{c}, z)$ such that $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = H(\mu \| w_1)$, where $\mu = H(M)$, $w_1 = \text{HighBits}(Az - ct,\ 2\gamma_2)$, and $c = \text{SampleInBall}(\tilde{c})$. It is important to note that c depends on $\tilde{c}$, $\tilde{c}$ depends on w_1, and w_1 depends on c, creating a circular dependency. Assuming that H has no weaknesses, the forger's best strategy appears to be to randomly select M and $w_1 \in R_q^k$, where each coefficient in w_1 is in the range $[0, (q-1)/2\gamma_2]$, and then compute $\mu = H(M)$, $\tilde{c} = H(\mu \| w_1)$, and $c = \text{SampleInBall}(\tilde{c})$. The remaining task is to find a z that satisfies the signature conditions.

Now, since $w_1 = \text{HighBits}(Az - ct,\ 2\gamma_2)$, we can write $Az - ct = 2\gamma_2 w_1 + w_0$ where $w_0 \in R_q^k$ with $-\gamma_2 < \text{coeff}(w_0) \le \gamma_2$. Rearranging terms gives $Az - w_0 = ct + 2\gamma_2 w_1$. Thus, the forger's task is to find a solution (z, w_0) to the linear system of equations

$$\begin{bmatrix} A \mid I_k \end{bmatrix} \begin{bmatrix} z \\ -w_0 \end{bmatrix} = ct + 2\gamma_2 w_1,$$

where $z \in R_q^\ell$ and $w_0 \in R_q^k$ are relatively small (more precisely, $\|z\|_\infty < \gamma_1 - \beta$ and $-\gamma_2 < \text{coeff}(w_0) \le \gamma_2$). This is an instance of the normal-form version of Inhomogeneous MSIS (see Exercises 15.34 and 15.35).

Dilithium has been proven to be existentially unforgeable under chosen-message attacks, assuming that the intractability of D-MLWE and MSIS, and that H behaves as a random function.

Full Dilithium. The Dilithium signature scheme standardized in FIPS 204 includes a method for compressing the public key component t, which requires modifications to the definitions of HighBits and LowBits, as well as the use of "hint bits". These changes lead to a small increase in signature size. The public key sizes for the ML-DSA-44, ML-DSA-65 and ML-DSA-87 parameter sets are 1312, 1952 and 2592 bytes, respectively, while the signature sizes are 2420, 3309 and 4627 bytes.

15.5 Chapter Endnotes

For an introduction to quantum computing, refer to the textbooks by Mermin [39] and Kaye, Laflamme and Mosca [25]. A more in-depth treatment is provided in Nielsen and Chuang's book *Quantum Computation and Quantum Information* [45]. The algorithms of Shor [48] and Grover [21] were groundbreaking advances in quantum computing. Jaques et al. [23] designed and analyzed quantum circuits for attacking AES using Grover's algorithm. While Grover's algorithm can be employed to find hash function collisions, Bernstein [5] convincingly argued that the classical van Oorschot-Wiener parallel collision search [50] remains more cost-effective. For an introduction to quantum-safe public-key cryptographic methods based on hash functions, error-correcting codes, lattices, and multivariate polynomial systems of equation, see the book edited by Bernstein, Buchmann and Dahmen [8].

A detailed survey of hash-based signature schemes is provided in Lafrance's thesis [27]. Lamport first proposed his one-time signature scheme to Diffie in 1975; it was described in Diffie and Hellman's 1976 paper on public-key cryptography [15], and later expanded in a 1979 technical report [28]. Winternitz's signature scheme and Merkle trees were detailed in Merkle's 1990 paper [38], though they were originally conceived in 1979. A formal security argument for the Winternitz signature scheme was given by Dods, Smart and Stam [16]. Bruinderink and Hülsing [11] and Fluhrer [19] examined the security of Winternitz's signature scheme when a key pair signs multiple messages, concluding that the scheme becomes insecure when standard parameters are used.

The Leighton-Micali signature scheme (LMS) was first described in a U.S. patent [30], and fully specified in IETF's RFC 8554 [36]. Katz [24] evaluated its security in the random oracle model, wherein the hash function H is treated as a public, randomly-selected function. Eaton [18] extended this analysis to the *quantum random oracle model*, where adversaries can query H in superposition.

The XMSS signature scheme was introduced by Buchmann, Dahmen and Hülsing [12], and later specified in IETF's RFC 8391 [22]. XMSS closely resembles LMS, with the main distinction being its use of "bitmasks"—pseudorandom bit strings that are XORed with inputs to the hash function. These bitmasks enable security proofs for XMSS in the "standard model", meaning the scheme's security does not rely on modeling the hash function as a random oracle.

Both LMS and XMSS, along with their hypertree variants (HSS and XMSS^{MT}), were later included in NIST's recommendation SP 800-208 [41]. SP 800-208 specifies parameter sets that employ SHA-256 or SHAKE256 as the underlying hash function, with 192- or 256-bit outputs. McGrew et al. [37] address practical challenges with managing state for stateful signature schemes like LMS and XMSS; see also the discussion in SP 800-208 [41].

SPHINCS+ was published in 2009 by Bernstein et al. [7], as an enhancement over the original SPHINCS [6]. Its complete specification can be found in NIST's FIPS 205 [44]. The few-times signature scheme HORS was proposed by Reyzin and Reyzin [47]. HORST was designed by Bernstein et al. [7]. SPHINCS+ uses a few-times signature scheme called

FORS ("Forest Of Random Subsets") [7] that improves upon HORST. The upper bound of equation (15.4) on the probability of a multicollision was derived by Suzuki et al. [49].

LWE was introduced in 2005 by Regev [46], who also provided a worst-case to average-case quantum reduction from approx-SIVP to LWE. Chatterjee et al. [13] offered a concrete analysis of Regev's reduction. A "ring" version of LWE, known as Ring-LWE, was first formulated by Lyubashevsky, Peikert and Regev [35], who applied tools from algebraic number theory to develop a worst-case to average-case quantum reduction from approx-SIVP in anti-cyclic lattices to Ring-LWE. Ring-LWE is a specialization of Module-LWE, where the matrix A of polynomials contains only one row. Module-LWE was first formulated by Brakerski, Gentry and Vaikuntanathan [10], and a worst-case to average-case quantum reduction for MLWE was presented by Langlois and Stehlé [29]. For a concrete analysis of the reductions for Ring-LWE and MLWE, see Koblitz et al. [26].

Kyber-PKE was developed by Bos et al. [9] and is based on the LWE-based public-key encryption scheme by Lindner and Peikert [31]. The Fujisaki-Okamoto transform was introduced in 1999 [20], and a variant of it used by Bos et al. [9] in the design of Kyber-KEM. For a comprehensive description of Kyber-KEM, including the specifications of the hash functions G, H, and J, the eXtendable Output Function (XOF) used to generate A, and the pseudorandom function (PRF) used to generate s, e, r, e_1 and e_2, see FIPS 203 [43].

SIS was introduced by Ajtai [1], who provided a worst-case to average-case reduction from approx-SIVP to SIS. Micciancio and Regev [40] improved the approximation factors in Ajtai's proof, and Bakos Lang [3] offered a concrete analysis of the improved proof. A "ring" version of SIS, known as Ring-SIS, was first formulated by Lyubashevsky and Micciancio [34]. Ring-SIS is a specialization of MSIS, where the matrix A of polynomials contains only one row. MSIS was formulated by Langlois and Stehlé [29], who also provided a worst-case to average-case reduction for MSIS.

Dilithium was designed by Ducas et al. [17] and draws on ideas from several earlier works, most notably those by Lyubashevsky [32, 33] and Bai and Galbraith [2]. It was heavily optimized to reduce public key and signature sizes, while ensuring fast and constant-time implementations. For a full description of Dilithium, including specifications of ExpandA, ExpandS, ExpandMask, H, SampleInBall, and the method for compressing the public-key component t, see FIPS 204 [42]. Formal reductionist security proofs for Dilithium are quite intricate; see Barbosa et al. [4] and Devevey et al. [14].

The dominant operation in Kyber and Dilithium is multiplication in the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Kyber uses $q = 3329$, whereas Dilithium uses $q = 2^{23} - 2^{13} + 1$; both employ $n = 256$. The classical algorithm for polynomial multiplication in R_q has running time $O(n^2)$ $\mathbb{Z}_q$-operations. The *Number-Theoretic Transform* (NTT), which is a finite field analogue of the Discrete Fourier Transform, provides a more efficient method for polynomial multiplication. Specifically, it enables multiplication in time $O(n \log n)$, offering a substantial reduction in complexity, both theoretically and practically, compared to the classical algorithm. To facilitate implementation of the NTT, the parameter n was selected to be a power of two, while the modulus q was selected to ensure divis-

ibility conditions: for Kyber, n divides $q-1$ (see [9, 43]), and for Dilithium, $2n$ divides $q-1$ (see [17, 42]). NTT is integral to the performance of both Kyber and Dilithium as described in the respective FIPS standards.

15.6 Exercises

Exercise 15.1 Let $H : \{0,1\}^m \longrightarrow \{0,1\}^n$ be a compression function, where $m \gg n$. Show that Grover's algorithm can find preimages for H using approximately $2^{n/2}$ quantum operations.

Exercise 15.2 Let $H : \{0,1\}^m \longrightarrow \{0,1\}^n$ be a compression function, where $m \gg n$. Show that Grover's algorithm can find second-preimages for H using approximately $2^{n/2}$ quantum operations.

Exercise 15.3 Show that Lamport's OTS is *insecure* if (a) H is not collision resistant; and (b) G is not preimage resistant. (c) Prove that Lamport's OTS is *secure* if H is collision resistant and G is preimage resistant.

Exercise 15.4 Suppose that Alice uses her key pair to sign three messages with the Lamport OTS, where $n = 256$. Suppose also than an adversary obtains the three signed messages. Show that the adversary can (with non-negligible probability) forge Alice's signature for messages of the adversary's choosing.

Exercise 15.5 Show that the Winternitz OTS is insecure if checksums are not used, i.e., the signature is $(s_0, s_1, \ldots, s_{\ell_1-1})$ instead of $(s_0, s_1, \ldots, s_{\ell-1})$.

Exercise 15.6 Suppose that a two-layer hypertree is employed with layer-1 tree of height $d_1 = 10$ and layer-2 trees of height $d_2 = 10$. Compare the signature size and signature verification time with the case where a Merkle tree with height $d = 20$ is employed. Both systems use the Winternitz OTS with parameters $n = 256$ and $w = 8$.

Exercise 15.7 Write pseudocode descriptions of key generation, signature generation, and signature verification for the Leighton-Micali signature scheme when used with a height-d Merkle tree.

Exercise 15.8 Write pseudocode descriptions of key generation, signature generation, and signature verification for HSS, when all employed Merkle trees have height d.

Exercise 15.9 Write pseudocode descriptions of key generation, signature generation, and signature verification for the HORST few-times signature scheme with parameters $n = 256$ and $k = 16$.

Exercise 15.10 Determine the sizes of public keys and signatures for the HORST few-times signature scheme with parameters $n = 256$ and $k = 16$.

Exercise 15.11 Consider a hash-based signature scheme that use virtual hypertrees. A virtual hypertree is represented as a hypertree with L layers, each layer consisting of Merkle trees of height d. Each leaf of the Merkle trees in layers 1 through $L-1$ is associated with an OTS public key, whereas each leaf of the layer-L Merkle trees is associated with an FTS public key. Write pseudocode description of key generation, signature generation, and signature verification for a stateless signature scheme that employs such a virtual hypertree.

Exercise 15.12 Write a program to find all solutions to the following LWE$(5, 3, 41, 2)$ instance:

$$A = \begin{bmatrix} 14 & 3 & 27 \\ 12 & 40 & 7 \\ 6 & 23 & 39 \\ 35 & 7 & 22 \\ 21 & 0 & 24 \end{bmatrix}, \quad b = \begin{bmatrix} 16 \\ 26 \\ 24 \\ 26 \\ 0 \end{bmatrix}.$$

Exercise 15.13 Prove that ss-LWE$(m, n, q, B) \leq$ LWE(m, n, q, B).

Exercise 15.14 Prove that LWE$(m, m, q, B) \leq$ ss-LWE$(m-n, n, q, B)$.

Exercise 15.15 Prove that ss-LWE and ss-DLWE are computationally equivalent.

Exercise 15.16 Let $R_q = \mathbb{Z}_{541}[x]/(x^5+1)$, and let $a(x) = 21 + 320x + 112x^2 + 73x^4 \in R_q$ and $b(x) = 539 + 300x + 23x^2 + 188x^3 + 432x^4$. Verify the following computations in R_q:
(a) $a(x) + b(x) = 19 + 79x + 135x^2 + 188x^3 + 505x^4$.
(b) $a(x) - b(x) = 23 + 20x + 89x^2 + 353x^3 + 182x^4$.
(c) $a(x) \cdot b(x) = 538 + 500x + 303x^2 + 388x^3 + 250x^4$.

Exercise 15.17 Let $u, v \in R_q = \mathbb{Z}_q[x]/(x^n+1)$. Prove that $\|uv\|_\infty \leq n\|u\|_\infty\|v\|_\infty$.

Exercise 15.18 The Euclidean norm of $z = (z_1, z_2, \ldots, z_n) \in \mathbb{R}^n$ is $\|z\|_2 = \sqrt{z_1^2 + z_2^2 + \cdots + z_n^2}$. Prove that $\|z\|_\infty \leq \|z\|_2 \leq \sqrt{n}\,\|z\|_\infty$ for all $z \in \mathbb{R}^n$.

Exercise 15.19 Write a program to find all solutions to the following MLWE$(3, 2, 4, 37, 1)$ instance:

$$A = \begin{bmatrix} 36 + 9x^3 & 24 + 6x + 29x^2 + 11x^3 \\ 7 + 14x + 15x^2 + 12x^3 & 27 + 9x + 22x^2 + 31x^3 \\ 19 + 10x + 3x^2 + 8x^3 & 20 + 23x + 12x^2 + 21x^3 \end{bmatrix}, \quad t = \begin{bmatrix} 25 + 30x + 20x^2 + 28x^3 \\ 1 + 15x + 19x^2 + 22x^3 \\ 6 + 30x + 18x^2 + x^3 \end{bmatrix}.$$

Exercise 15.20 Let $q = 173$. Verify that $\text{Round}_q(40+47x+157x^2+80x^3+160x^4) = x+x^3$.

Exercise 15.21 Consider an instance of Kyber-PKE with domain parameters $q = 197$, $n = 4$, $k = 3$, $\eta_1 = 2$ and $\eta_2 = 2$. Bob's encryption key is (A, t) where

$$A = \begin{bmatrix} 20 + 189x + 77x^2 + 196x^3 & 96 + 150x + 146x^2 + 28x^3 & 88 + 59x + 195x^2 + 57x^3 \\ 177 + 64x + 192x^2 + 99x^3 & 121 + 10x + 21x^2 + 94x^3 & 142 + 26x + 173x^2 + 195x^3 \\ 185 + 7x + 86x^2 + 176x^3 & 33 + 57x + 76x^2 + 21x^3 & 133 + 151x + 196x^2 + 117x^3 \end{bmatrix}$$

and

$$t = \begin{bmatrix} 135 + 49x + 22x^2 + 149x^3 \\ 97 + 129x + 186x^2 + 107x^3 \\ 59 + 21x + 102x^2 + 80x^3 \end{bmatrix},$$

and his decryption key is

$$s = \begin{bmatrix} 1 - x - 2x^2 - 2x^3 \\ -2 + 2x - 2x^2 - x^3 \\ 2 - 2x + 2x^2 + 2x^3 \end{bmatrix}.$$

To encrypt the plaintext $m = 1010$ for Bob, Alice selects

$$r = \begin{bmatrix} -2 + 2x - 2x^2 + x^3 \\ 1 - x + 2x^2 + 2x^3 \\ 2 + 2x + 2x^3 \end{bmatrix}, \quad e_1 = \begin{bmatrix} 1 - x + x^3 \\ -2 - 2x \\ -x - x^2 + x^3 \end{bmatrix} \quad \text{and} \quad e_2 = 1 + 2x + 2x^2.$$

(a) Verify that the ciphertext that Alice computes is $c = (u, v)$ where

$$u = \begin{bmatrix} 90 + 84x + 106x^2 + 97x^3 \\ 152 + 78x + 77x^2 + 36x^3 \\ 104 + 88x + 192x^2 + 118x^3 \end{bmatrix} \text{ and } v = 2 + 49x + 56x^2 + 47x^3.$$

(b) Verify that decryption of c by Bob will recover m.

Exercise 15.22 Show that Kyber-PKE is insecure against chosen-ciphertext attacks.

Exercise 15.23 (Kyber encryption key size) Consider the ML-KEM-768 parameters $q = 3329$, $n = 256$, $k = 3$.
(a) Show that a Kyber encryption key (A, t) has size 4,608 bytes.
(b) Suppose now that A is generated from a randomly selected 256-bit seed ρ using a (publicly-known) pseudorandom bit generator. Then (ρ, t) can be used as the Kyber encryption key. Show that this modified key has size 1,184 bytes.

Exercise 15.24 Prove Fact 15.25.

Exercise 15.25 Consider Kyber public-key encryption with ciphertext compression as described at the end of §15.3.4. Define the compression errors $e_u = u' - u$ and $e_v = v' - v$. Prove that the decryption error polynomial $v' - s^T u' - \lceil q/2 \rfloor m$ is equal to $e^T r + e_2 - s^T e_1 + e_v - s^T e_u$.

Exercise 15.26 (Kyber ciphertext size) Consider the ML-KEM-768 Kyber parameters (Table 15.1). (a) Show that if ciphertext compression is not used, then a ciphertext has size 1,536 bytes. (b) Show that a compressed ciphertext has size 1,088 bytes.

Exercise 15.27 Write a program to find all solutions to the following SIS(3, 5, 17, 3) instance:

$$A = \begin{bmatrix} 1 & 12 & 9 & 0 & 4 \\ 2 & 13 & 3 & 14 & 12 \\ 1 & 8 & 16 & 5 & 1 \end{bmatrix}.$$

Exercise 15.28 The $\text{SIS}_2(n, m, q, B)$ problem is the following: given $A \in_R \mathbb{Z}_q^{n\times m}$, find $z \in \mathbb{Z}^m$ such that $Az = 0 \pmod q$ where $0 < \|z\|_2 \leq B$. Prove that $\text{SIS}(n, m, q, B) \leq \text{SIS}_2(n, m, q, B) \leq \text{SIS}(n, m, q, B/\sqrt{m})$.

Exercise 15.29 Let $A \in_R \mathbb{Z}_q^{n\times m}$, where $m > n \log q$. Define the hash function $H_A : \{0,1\}^m \longrightarrow \mathbb{Z}_q^n$ by $H_A(z) = Az \pmod q$. Prove that H_A is collision resistant assuming that $\text{SIS}(n, m, q, 1)$ is hard.

Exercise 15.30 Prove that nf-ISIS(n, m, q, B) and ISIS$(n, m + n, q, B)$ are computationally equivalent.

Exercise 15.31 Prove that ss-LWE$(m, n, q, B) \leq$ ISIS$(m, m + n, q, B)$.

Exercise 15.32 Write a program to find all solutions to the following MSIS$(2, 3, 4, 71, 10)$ instance:

$$A = \begin{bmatrix} 45 + 45x^2 + 17x^3 & 38 + 69x + 33x^2 + 2x^3 & 14 + 5x + 21x^2 + 41x^3 \\ 3 + 29x + 32x^2 + 45x^3 & 52 + 60x + 69x^3 & 49 + 11x + 55x^2 + 70x^3 \end{bmatrix}.$$

Exercise 15.33 Let $q = 7937$, $\alpha = 496$, and $a(x) = 5871 + 1534x + 6660x^2 + 3609x^3 + 17370x^4 \in \mathbb{Z}_q[x]$. Verify that $\text{HighBits}(a(x), \alpha) = 12 + 3x + 13x^2 + 7x^3 + 4x^4$ and $\text{LowBits}(a(x), \alpha) = -81 + 46x + 212x^2 + 137x^3 - 247x^4$.

Exercise 15.34 The *Module Inhomogeneous Short Integer Solutions* problem MISIS(k, ℓ, n, q, B) is the following. Given $A \in R_q^{k\times\ell}$ and $b \in_R R_q^k$, find $z \in R_q^\ell$ such that $Az = b$ where $\|z\|_\infty \leq B$. Prove that MSIS $\leq$ MISIS and MISIS $\leq$ MSIS.

Exercise 15.35 The *normal-form MISIS* problem nf-MISIS(k, ℓ, n, q, B) is the following. Given $A \in_R R_q^{k\times\ell}$ and $b \in_R R_q^k$, find $z \in R_q^{k+\ell}$ such that $[A \mid I_k]z = b$ and $\|z\|_\infty \leq B$. Here, I_k is the $k \times k$ identity matrix over R_q. Prove that nf-MISIS$(k, \ell, n, q, B) \leq$ MISIS$(k, k + \ell, n, q, B)$ and MISIS$(k, k + \ell, n, q, B) \leq$ nf-MISIS(k, ℓ, n, q, B).

Exercise 15.36 Describe and analyze the FIPS 204 SampleInBall procedure for generating pseudorandom polynomials in B_τ. This procedure is based on the "Fisher-Yates shuffle".

Exercise 15.37 Describe the FIPS 204 method for compressing the Dilithium public-key component t. This method utilizes the Power2Round, MakeHint, and UseHint procedures, in addition to slightly modifying the HighBits and LowBits functions. See also [17].

Bibliography

[1] M. Ajtai. Generating hard instances of lattice problems. In *Symposium on Theory of Computing (STOC)*, pages 99–108, 1996.

[2] S. Bai and S. Galbraith. An improved compression technique for signatures based on learning with errors. In *The Cryptographers' Track at RSA Conference (CT-RSA)*, pages 28–47, 2014.

[3] E. Bakos Lang. *Worst-Case to Average-Case Reductions for the SIS Problem: Tightness and Security*. MMath thesis, University of Waterloo (Canada), 2019.

[4] M. Barbosa, G. Barthe, C. Doczkal, J. Don, S. Fehr, B. Grégoire, Y. Huang, A. Hülsing, Y. Lee, and X. Wu. Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In *CRYPTO*, pages 358–389, 2023.

[5] D. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?, 2009. In: Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems, available from https://cr.yp.to/hash/collisioncost-20090823.pdf.

[6] D. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In *EUROCRYPT*, pages 368–397, 2015.

[7] D. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, and J. Rijneveld. The SPHINCS+ signature framework. In *ACM Comp. & Comm. Security (CCS)*, pages 2129–2149, 2019.

[8] D. Bernstin, J. Buchmann, and E. Dahmen. *Post-Quantum Cryptography*. Springer, 2009.

[9] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *IEEE Symp. Security and Privacy*, pages 353–367, 2018.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 7(3), 2014.

[11] L. Bruinderink and A. Hülsing. "Oops, I did it again" — Security of one-time signatures under two-message attacks. In *Workshop on Selected Areas in Cryptography (SAC)*, pages 299–322, 2018.

[12] J. Buchmann, E. Dahmen, and A. Hülsing. XMSS — a practical forward secure signature scheme based on minimal security assumptions. In *Post-Quantum Cryptography (PQCrypto)*, pages 117–129, 2011.

[13] S. Chatterjee, N. Koblitz, A. Menezes, and P. Sarkar. Another look at tightness II: Practical issues in cryptography. In *Paridigms in Cryptography (Mycrypt)*, pages 21–55, 2017.

[14] J. Devevey, P. Fallahpour, A. Passelègue, D. Stehlé, and K. Xagawa. A detailed analysis of Fiat-Shamir with abort. In *CRYPTO*, pages 327–357, 2023.

[15] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22:644–654, 1976.

[16] C. Dods, N. Smart, and M. Stam. Hash based digital signature schemes. In *Cryptography and Coding*, pages 96–115, 2005.

[17] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, and G. Seiler. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[18] T. Eaton. Leighton-Micali hash-based signatures in the quantum random-oracle model. In *Workshop on Selected Areas in Cryptography (SAC)*, pages 263–280, 2018.

[19] S. Fluhrer. Oops, I did it again revisited: Another look at reusing one-time signatures, 2023. IACR ePrint, http://eprint.iacr.org/2023/1905.

[20] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999.

[21] L. Grover. A fast quantum mechanical algorithm for database search. In *Symposium on Theory of Computing (STOC)*, pages 212–219, 1996.

[22] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. RFC 8391—XMSS: eXtended Merkle Signature Scheme, May 2018. Informational. IETF.

[23] S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia. Implementing Grover oracles for quantum key search on AES and LowM. In *EUROCRYPT*, pages 280–310, 2020.

[24] J. Katz. Analysis of a proposed hash-based signature standard. In *Security Standardisation Research (SSR)*, pages 261–273, 2016.

[25] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2006.

[26] N. Koblitz, S. Samajder, P. Sarkar, and S. Singha. Concrete analysis of approximate ideal-SIVP to decision ring-LWE reduction. *Advances in Mathematics of Communicaitons*, 18:1216–1258, 2024.

[27] P. Lafrance. *Digital Signature Schemes based on Hash Functions.* MMath thesis, University of Waterloo (Canada), 2017.

[28] L. Lamport. Constructing digital signatures from a one way function, 1979. Technical Report CSL-98, SRI International.

[29] A. Langlois and D. Stehlé. Worst-case to average-case reduction for module lattices. *Designs, Codes and Cryptography*, 75:565–599, 2015.

[30] T. Leighton and S. Micali. Large provably fast and secure digital signature schemes based on secure hash functions, 1995. U.S. patent 5,432,852.

[31] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In *The Cryptographers' Track at RSA Conference (CT-RSA)*, pages 319–339, 2011.

[32] V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, pages 598–616, 2009.

[33] V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, pages 738–716, 2012.

[34] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP*, pages 144–155, 2006.

[35] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6), 2013.

[36] D. McGrew, M. Curcio, and S. Fluhrer. RFC 8554—Leighton-Micali hash-based signatures, Apr. 2019. Informational. IETF.

[37] D. McGrew, P. Kampanakis, S. Fluhrer, S. Gazdag, D. Butin, and J. Buchmann. State management for hash-based signatures. In *Security Standardisation Research (SSR)*, pages 244–260, 2016.

[38] R. Merkle. A certified digital signature. In *CRYPTO*, pages 218–238, 1990.

[39] N. D. Mermin. *Quantum Computer Science: An Introduction.* Cambridge University Press, 2007.

[40] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Computing*, 37:267–302, 2007.

[41] National Institute of Standards and Technology. Recommendation for stateful hash-based signature schemes, 2020. SP 800-208.

[42] National Institute of Standards and Technology. Module-lattice-based digital signature standard, 2024. FIPS 204.

[43] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard, 2024. FIPS 203.

[44] National Institute of Standards and Technology. Stateless hash-based digital signature standard, 2024. FIPS 205.

[45] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2nd edition, 2011.

[46] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009.

[47] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Australasian Conf. on Info. Security & Privacy (ACISP)*, pages 144–153, 2002.

[48] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 421:303–332, 1999.

[49] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota. Birthday paradox for multi-collisions. In *Information Security and Cryptology (ICISC)*, pages 29–40, 2006.

[50] P. van Oorschot and M. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999.