A Tutorial on Post-quantum Cryptography

Douglas R. Stinson

David R. Cheriton School of Computer Science
University of Waterloo

August 11, 2025

1. **Introduction**
   - Goals and summary
   - Introduction to quantum computing
   - Order-finding problem
   - Period-finding
   - NIST standardization of post-quantum cryptography

Goals

- We begin with a short introduction to quantum computing and its potential impact on current cryptographic algorithms.
- Then we summarize the ongoing NIST standardization process for post-quantum cryptography.
- The main part of the lectures is a discussion of approaches to post-quantum cryptography, emphasizing the underlying mathematical techniques. These include:
  - **hash-based signature schemes** (e.g., *SPHINCS+*)
  - **code-based cryptography** (e.g., *McEliece*, *Niederreiter*, *BIKE*, *HQC*)
  - **lattice-based cryptography** (e.g., *NTRU*, *Regev*, *Kyber*, *Dilithium*)
  - **multivariate cryptography** (e.g., *Oil and Vinegar*)
- We assume a basic background in cryptography, algebra and number theory.

What is Quantum Computing?

From the IBM web page on quantum computing:

> *While classical computers rely on binary bits (zeros and ones) to store and process data, quantum computers can encode even more data at once using quantum bits, or qubits, in superposition.*
>
> *A qubit can behave like a bit and store either a zero or a one, but it can also be a weighted combination of zero and one at the same time. When combined, qubits in superposition can scale exponentially. Two qubits can compute with four pieces of information, three can compute with eight, and four can compute with sixteen.*
>
> *However, each qubit can only output a single bit of information at the end of the computation. Quantum algorithms work by storing and manipulating information in a way inaccessible to classical computers, which can provide speedups for certain problems.*

Introduction to Quantum Computing (cont.)

- The basic idea of quantum computing dates back to at least 1980.
- The relevance of quantum computing to cryptography became evident with the publication of *Shor's Algorithm* in 1994.
- However, the development of a practical quantum computer appears to be some years in the future.
- Despite intense research during the last $30+$ years, construction of a scalable, fault-tolerant quantum computer has not been achieved yet.
- Experts have expressed various opinions as to how soon a quantum computer would be able to factor a 2048-bit RSA modulus. In the "Quantum Threat Timeline Report 2024, Executive Summary," various "optimistic" and "pessimistic" estimates are provided.
- For example, the estimated probability that a quantum computer will be built in the next five years that will be able to a 2048-bit RSA modulus ranges from 5% to 14%. On the other hand, for a twenty-year window, the estimated probabilities are 60% (pessimistic) and 82% (optimistic).

An IACR eprint

The following paper is an amusing commentary on some exaggerated claims relating to factorizations by quantum computers.

Replication of Quantum Factorisation Records with an 8-bit Home Computer, an Abacus, and a Dog

Peter Gutmann, University of Auckland
pgut001@cs.auckland.ac.nz

Stephan Neuhaus, Zürcher Hochschule für Angewandte Wissenschaften
neut@zhaw.ch

March 2025

1 Abstract

This paper presents implementations that match and, where possible, exceed current quantum factorisation records using a VIC-20 8-bit home computer from 1981, an abacus, and a dog. We hope that this work will inspire future efforts to match any further quantum factorisation records, should they arise.

Potential Impact of Quantum Computing

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

Potential Impact of Quantum Computing (cont.)

- Consider a hash function that creates an n-bit output (message digest), so there are 2^n possible message digests.
- Classical algorithms for **Preimage** and **Second Preimage** have complexity $O(2^n)$.
- A classical birthday attack for **Collision** has complexity $O(2^{n/2})$.
- Quantum algorithms (based on *Grover's algorithm*) for **Preimage** and **Second Preimage** have complexity $O(2^{n/2})$.
- The lowest complexity for a quantum algorithm solving **Collision** is $O(2^{n/3})$.
- This algorithm, which is due to Brassard, Hoyer and Tapp, is based on *Grover's algorithm*.

Post-quantum Cryptography

- Much current research is being carried out to potential ways of constructing public-key cryptosystems based on different computational problems, which hopefully would not be susceptible to attacks carried out by quantum computers.
- The term post-quantum cryptography is used to describe such cryptographic schemes.
- This phrase is mainly used in conjunction with public-key encryption (especially Key Encapsulation Mechanisms) and signature schemes.
- NIST initiated an ongoing standardization process for post-quantum cryptography in 2016 (more details later).

Key Encapsulation Mechanisms

- a key encapsulation mechanism (or KEM) generalizes the familiar ideas of key agreement and hybrid cryptography.
- We provide a simple illustration using *RSA* public-key encryption.
- Suppose Alice wants to use *RSA* to encrypt a (random) 128-bit *AES* key, say K, to send to Bob.
- In a traditional example of hybrid cryptography, K would be used to encrypt messages sent between Alice and Bob in a subsequent session.
- *RSA* requires a 2048-bit plaintext, so K would have to be padded to 2048 bits before it is encrypted.
- An alternative approach is to generate a random 2048-bit plaintext, say x and encrypt it using *RSA*.
- Alice and Bob will both use a suitable public key derivation function (typically based on a hash function) to derive K from x.

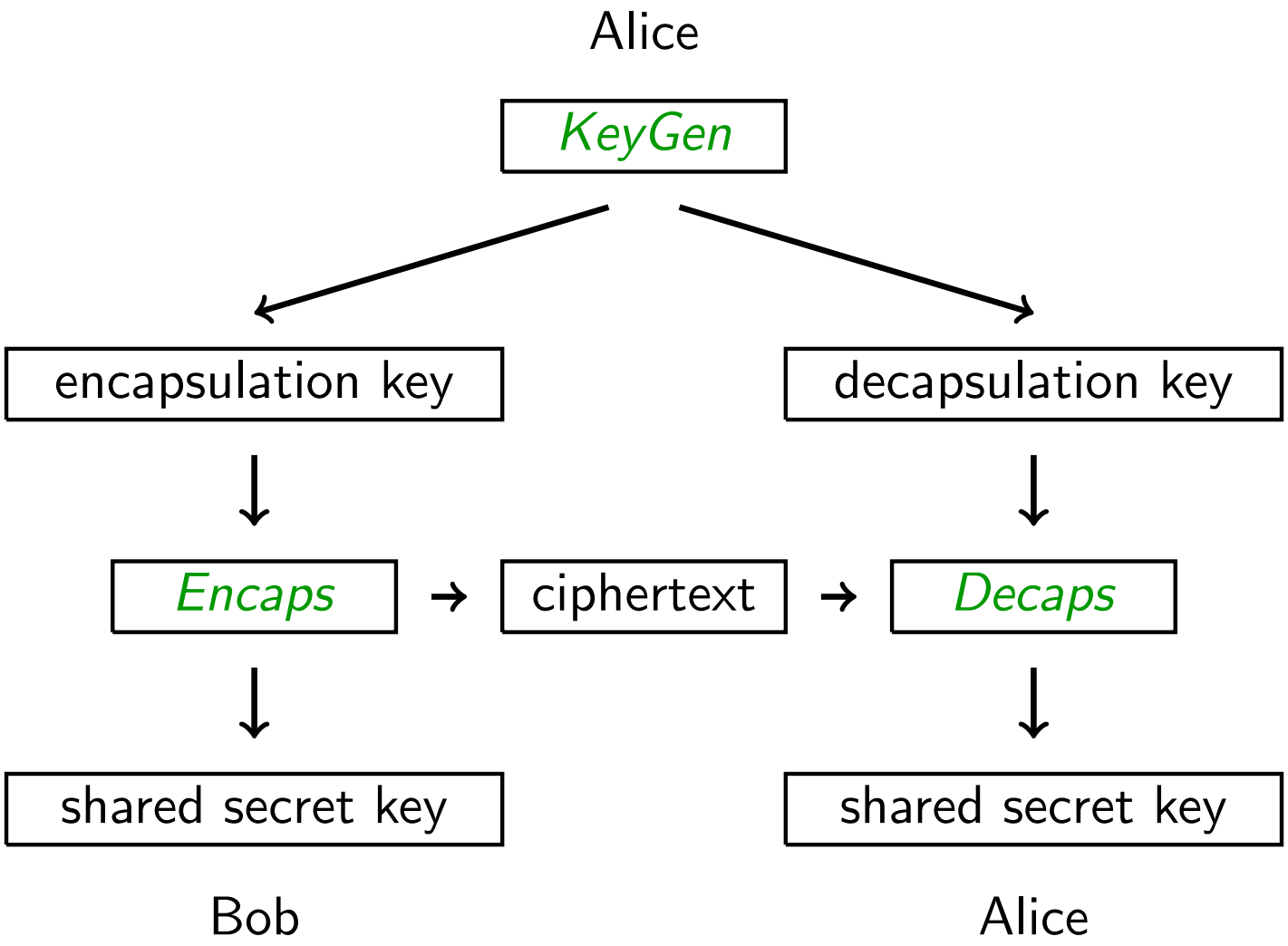


Figure: Key encapsulation mechanism

Techniques for Post-quantum Cryptography

There have been several interesting approaches to post-quantum cryptography, including the following:

- **lattice-based cryptography:** *NTRU* is defined using arithmetic in certain polynomial rings. Many examples of lattice-based cryptography are based on the **Learning With Errors** problem.
- **code-based cryptography:** The *McEliece Cryptosystem* involves error-correcting codes (specifically, Goppa codes).
- **multivariate cryptography:** Multivariate cryptography includes cryptosystems such as *Hidden Field Equations*, as well as signature schemes such as *Oil and Vinegar*.
- **hash-based cryptography:** Hash-based cryptography (e.g, *Lamport One-time Signatures*) is used primarily for signature schemes.
- **isogeny-based cryptography:** Isogeny-based cryptography (e.g., *SIKE*, which has been broken) uses morphisms between different elliptic curves.

Timeline for Post-quantum Cryptography

- The proposed techniques for post-quantum cryptography are not proven to be immune to attacks by quantum computers.
- The current approach is to utilize problems that, at present, are not susceptible to quantum attacks based on currently known algorithms.
- Even 15 years of lead time to develop post-quantum cryptographic algorithms leaves little margin for delay, since the development, standardization, and deployment of new cryptographic technologies takes a considerable amount of time.
- Thus, implementation of post-quantum cryptography is viewed by many as a serious problem of immediate and pressing concern.
- In 2015, the NSA announced its intention to transition to post-quantum cryptography.
- The first three NIST standards (one encryption scheme and two signature schemes) were published in 2024 (more detail about the NIST standardization process will be given a bit later).

Order-finding Problem

Shor's factoring algorithm can easily be described assuming an oracle for the **order-finding** problem. We define this problem now.

Order-finding

Instance: A positive integer n, and an element $a \in {\mathbb{Z}_n}^*$.
Find: The order of a in the multiplicative group $({\mathbb{Z}_n}^*, \cdot)$.

The order of a is the smallest positive integer t such that $a^t \equiv 1 \bmod n$. This integer is denoted by $ord(a)$. It can easily be proven that $ord(a)$ is a divisor of $\phi(n) = |{\mathbb{Z}_n}^*|$.

Square Roots of 1 modulo n

- The following is similar to the analysis of the *Rabin Cryptosystem*.
- Suppose that $n = pq$, where p and q are distinct odd primes.
- Suppose that $b^2 \equiv 1 \pmod{n}$ and $b \not\equiv \pm 1 \pmod{n}$.
- Note that 1 has four square roots in $\mathbb{Z}_n$, so there are two square roots for which the above conditions hold.
- Then $\gcd(b-1, n)$ and $\gcd(b+1, n)$ yield the factors p and q.
- If an element $a \in {\mathbb{Z}_n}^*$ has order r, where r is even, then $b = a^{r/2} \bmod n$ is a square root of 1 modulo n.
- If a is chosen randomly, then there is a good probability that $ord(a)$ is even.
- Further, if $ord(a)$ is even, then we might expect that there is good chance that $b \not\equiv \pm 1 \pmod{n}$; in this case, n can be factored by the $\gcd$ computation mentioned above.

Shor's Quantum Factoring Algorithm

Shor's factoring algorithm is a randomized Las Vegas algorithm.

1. Choose a random integer a, where $1 < a < n$.
2. If $\gcd(a, n) > 1$, then we can factor n and QUIT (success).
3. Otherwise, use a quantum algorithm to solve the instance of **Order-finding** consisting of n and a. The output of this algorithm is the integer $r = \mathit{ord}(a)$.
4. If r is odd, then QUIT (failure).
5. (Here r is even.) Compute $b = a^{r/2} \bmod n$.
6. If $b \equiv -1 \pmod{n}$, then QUIT (failure).
7. Compute $\gcd(b-1, n)$ and $\gcd(b+1, n)$ to factor n.

In the case of failure, we just repeat the algorithm with another random choice of a.

Quantum Discrete Log Algorithm

- *Shor's algorithm* for the **Discrete Log** problem can be discussed assuming an oracle for the **period-finding** problem.
- A function f defined on an additive abelian group G is periodic with period d if $f(x) = f(x + d)$ for all $x \in G$.
- The value d is required to be a non-zero element of G.
- Note that the period of a function f is not defined uniquely.
- For example, if d is a period, then $d + d$ is also a period (provided that $d + d \neq 0$).

Period-finding

Instance: A periodic function f defined on an additive abelian group G.

Find: A period d for the function f.

Quantum Discrete Log Algorithm (cont.)

- Suppose $g, a \in \mathbb{Z}_p^*$ are specified and we want to find x such that $g^x = a$.
- Define the function f to be $f(\alpha, \beta) = g^\alpha a^{-\beta} \bmod p$, where $\alpha, \beta \in \mathbb{Z}_{p-1}$.
- A period of f will be an ordered pair (d_1, d_2) such that $f(\alpha, \beta) = f(\alpha + d_1, \beta + d_2)$ for all α, β.
- We observe the following:

$$\begin{aligned} f(\alpha, \beta) = f(\alpha + d_1, \beta + d_2) &\Leftrightarrow g^\alpha a^{-\beta} \equiv g^{\alpha + d_1} a^{-\beta - d_2} \pmod{p} \\ &\Leftrightarrow a^{d_2} \equiv g^{d_1} \pmod{p} \\ &\Leftrightarrow (g^x)^{d_2} \equiv g^{d_1} \pmod{p} \\ &\Leftrightarrow x d_2 \equiv d_1 \pmod{p-1} \\ &\Leftrightarrow x \equiv (d_2)^{-1} d_1 \pmod{p-1}. \end{aligned}$$

- Thus, given a period (d_1, d_2) of f, we can compute $x = \log_g a$ whenever d_2 is invertible modulo $p - 1$.

NIST Standardization Process–Round 1

- A Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms was published on Dec. 20, 2016.
- The deadline for submissions was Nov. 30, 2017.
- 23 signature schemes and 59 encryption/KEM schemes were submitted.
- 69 of the 82 submissions were approved by NIST to participate in Round 1.
- These included lattice, code-based, hash-based, multivariate, braid group, isogeny-based, and "miscellaneous" schemes.

NIST Standardization Process–Round 1 (cont.)

Three evaluation criteria were specified:

1. **Security:** General-use encryption schemes were required to be semantically secure against adaptive chosen-ciphertext attack. For ephemeral use cases, semantic security against chosen plaintext attack sufficed. Signature schemes were expected to be existentially unforgeable against adaptive chosen message attack.
2. **Cost and performance:** This includes factors such as size of public keys, ciphertext and signatures; efficiency of key generation, public and private key operations; and probability of decryption failures.
3. **Algorithm and implementation characteristics:** This includes flexibility of algorithms WRT a wide variety of platforms, simplicity of design, IP issues, etc.

Security Categories

NIST also defined five security categories: Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for the following attack scenarios:

1. key search on a block cipher with a 128-bit key (e.g. *AES128*)
2. collision search on a 256-bit hash function (e.g. *SHA256/ SHA3-256*)
3. key search on a block cipher with a 192-bit key (e.g. *AES192*)
4. collision search on a 384-bit hash function (e.g. *SHA384/ SHA3-384*)
5. key search on a block cipher with a 256-bit key (e.g. *AES 256*)

NIST Standardization Process–Round 2

- It was announced on Jan. 30, 2019, that 26 of the 69 schemes would advance to Round 2.
- There were 17 encryption schemes and 9 signature schemes chosen.
- They can be categorized as follows:

1. **lattice-based encryption:** Crystals-Kyber, FrodoKEM, LAC, NewHope, NTRU, NTRU Prime, Round5, Saber, Three Bears.
2. **code-based encryption:** Classic McEliece, NTS-KEM, BIKE, HQC, LEDAcrypt, Rollo, RQC.
3. **isogeny-based encryption:** SIKE.
4. **lattice-based signature:** Crystals-Dilithium, Falcon, qTESLA.
5. **multivariate signature:** GeMSS, LUOV, MQDSS, Rainbow.
6. **hash-based signature:** Picnic, SPHINCS+.

NIST Standardization Process–Round 3

- On July 22, 2020, NIST chose 15 candidates to advance to Round 3.
- Seven candidates were designated as finalists and the other eight were alternate algorithms.
- The seven finalists were *Classic McEliece* (code-based encryption); *Crystals-Kyber*, *NTRU* and *Saber* (lattice-based encryption); *Crystals-Dilithium* and *Falcon* (lattice-based signature); and *Rainbow* (multivariate signature).
- The eight alternate candidates were *BIKE* and *HQC* (code-based encryption); *FrodoKEM* and *NTRU Prime* (lattice-based encryption); *SIKE* (isogeny-based encryption); *GeMSS* (multivariate signature); *Picnic* and *SPHINCS+* (hash-based signature).
- The seven finalists were thought by NIST to be "ready for standardization soon."
- The eight alternate candidates were regarded as "potential candidates for future standardization."

Standardization and Round 4

- Four "selected algorithms" were announced on July 5, 2022:
  - *CRYSTALS-Kyber* (lattice-based PKE/KEM; FIPS 203, 2024)
  - *CRYSTALS-Dilithium* (lattice-based signature; FIPS 204, 2024)
  - *FALCON* (lattice-based signature)
  - *SPHINCS+* (hash-based signature; FIPS 205, 2024)
- Also, four additional candidates (all PKE/KEM) were announced for Round 4:
  - *BIKE*, *Classic McEliece*, *HQC* (all code-based)
  - *SIKE* (isogeny-based) – however, it was broken in August 2022
- *HQC* was selected for standardization on March 11, 2025.

Comments

- There is a lack of diversity in the chosen candidates: three lattice-based, one code-based and one hash-based.
- No multivariate schemes survived, and the only isogeny-based scheme suffered a fatal attack.
- In September 2022, NIST called for additional digital signature proposals to be considered in the PQC standardization process.
- Submission packages were due by June 1, 2023.
- There were 40 submissions that were regarded as complete; they proceeded to Round 1.
- In October 2024, NIST selected 14 candidate algorithms to move forward to Round 2.
- The 14 proposed schemes are based on a variety of mathematical techniques, including schemes that are code-based, lattice-based, multivariate schemes, MPC-in-the-head, isogeny-based and symmetric cryptography-based schemes (more details later).

2 Hash-based signature schemes

- Hash functions
- Lamport Signature Scheme
- Winternitz Signature Scheme
- Merkle trees
- Multilevel Merkle trees

Hash Functions

- Various signature schemes can be built from cryptographic hash functions.
- We assume that we have a hash function that is preimage resistant, second preimage resistant and collision resistant.
- *SHA3-224* might be a reasonable choice.
- The first construction we give is a secure one-time signature scheme built from a from a preimage resistant hash function, AKA a one-way function.
- A signature scheme is a one-time signature scheme if it is secure when only one message is signed. The signature can be verified an arbitrary number of times, of course.
- We begin with a classic construction called the *Lamport Signature Scheme*, which was published in 1979.

Lamport Signature Scheme

Let k be a positive integer, let $\mathcal{P} = \{0,1\}^k$ and let $\mathcal{A} = Y^k$. (We will sign one k-bit message.) Suppose $f : Y \to Z$ is a one-way function.

Let $y_{i,j} \in Y$ be chosen at random (for $1 \leq i \leq k$, $j = 0, 1$) and let $z_{i,j} = f(y_{i,j})$, for all i, j.

The key K consists of the $y_{i,j}$'s (preimages) and the $z_{i,j}$'s (images). The $y_{i,j}$'s are the private key while the $z_{i,j}$'s are the public key.

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

$$sig_K(x_1, \ldots, x_k) = (y_{1,x_1}, \ldots, y_{k,x_k}).$$

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

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

Discussion

- Informally, this is how the *Lamport Signature Scheme* works.
- A message to be signed is a binary k-tuple, where the value of k is fixed ahead of time.
- Each bit of the message is signed individually.
- If the ith bit of the message equals j (where $j \in \{0, 1\}$), then the ith element of the signature is the value $y_{i,j}$, which is a preimage of the public key value $z_{i,j}$.
- The verification consists simply of checking that each element in the signature is a preimage of the public key element $z_{i,j}$ that corresponds to the ith bit of the message.
- This can be done using the public function f.
- To sign a k-bit message, we require a very large public key, consisting of $2k$ values $z_{i,j}$ from the set Z. Since these z-values are probably outputs of a secure hash function, they would each be least 224 bits in length (for example, if we used the hash function *SHA3-224*).

Example

We pretend that $f(y) = 3^y \bmod 7879$ is a one-way function. Suppose $k = 3$, and Alice chooses the six (secret) random numbers to be her private key:

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

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

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

These z's are the public key.
Now, suppose Alice wants to sign the message $x = (1, 1, 0)$. The signature for x is $(y_{1,1}, y_{2,1}, y_{3,0}) = (735, 2467, 4285)$. To verify this signature, it suffices to compute the following:

$$f(735) = 3810, \quad f(2467) = 4721, \quad f(4285) = 268.$$

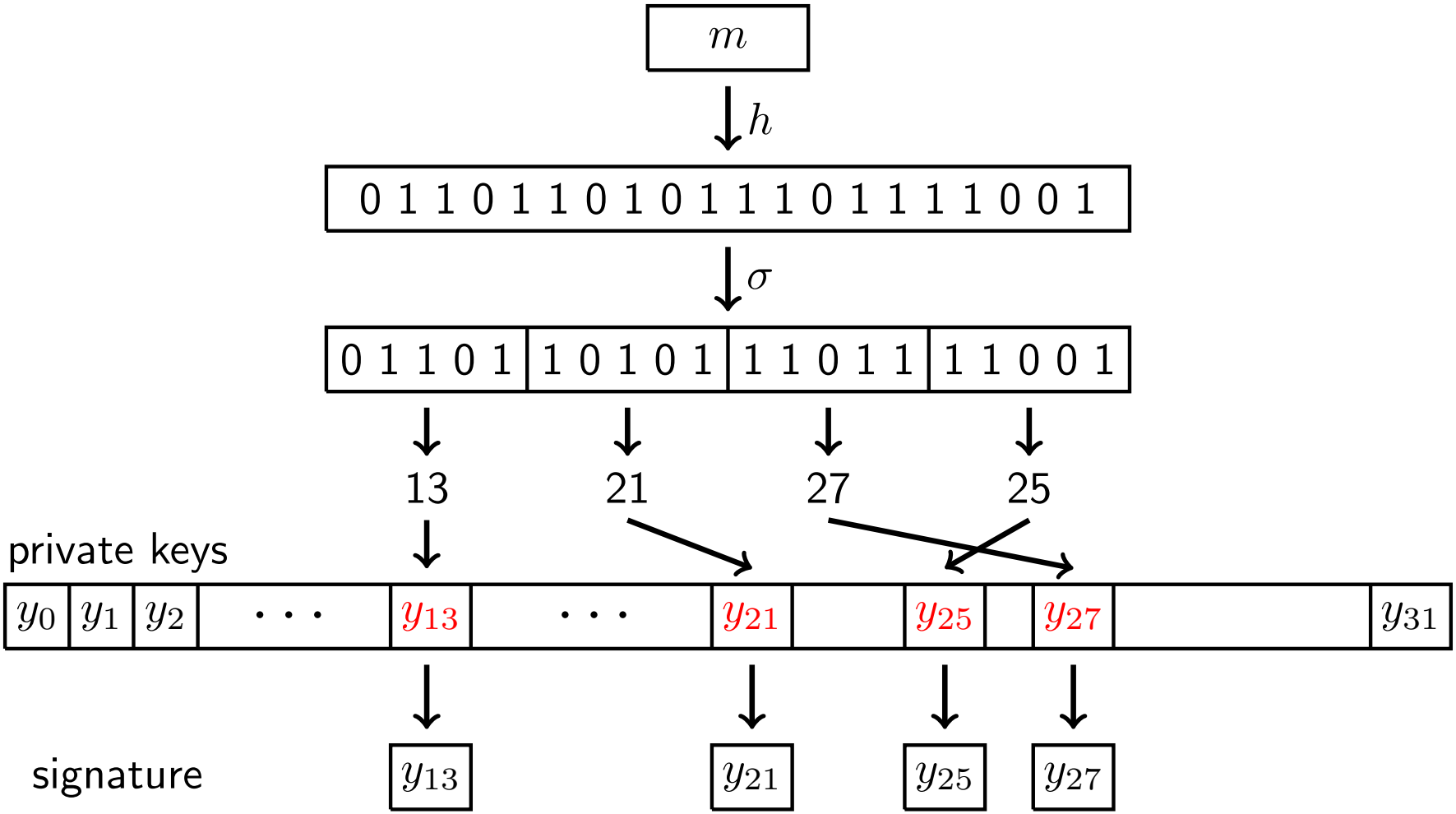
Few-times Signature Scheme

- If an adversary sees one message and its signature, then they will be unable to forge a signature on a second message.
- However, given signatures on two different messages, it is easy for an adversary to construct signatures for another message different from the first two (unless the first two messages differ in exactly one bit).
- The *HORS* (*hash to obtain random subset*) scheme creates a signature scheme that can be re-used a few times.
- This idea is due to Reyzin and Reyzin in 2002.
- We use the same framework as the *Lamport Scheme*.
- Assume that $f : Y \to Z$ is a one-way function (e.g., a hash function).
- Now we have a pool of t public keys, denoted $z_1, \ldots, z_t$.
- We also have t corresponding private keys (preimages), denoted $y_1, \ldots, y_t$, where $f(y_i) = z_i$ for $1 \leq i \leq t$.

Few-times Signature Scheme (cont.)

- Suppose we want to sign message m.
- We compute $h(m)$ and split the result into a set of κ integers in the set $\{1, \ldots, t\}$.
- Here h is a hash function that is being used to select a subset of the t keys.
- Specifically, h determines a selection function σ that maps a message m to a κ-subset of $\{1, \ldots, t\}$.
- The security of *HORS* depends on the cover-free properties of the selection function σ.
- In order to verify the signature, all κ components of the signature need to be verified using the corresponding public keys.
- This process is illustrated for $\kappa = 4$ and $t = 32$ in the example on the next slide, which is taken from David Wong's blog.

HORS Few-times Signature Scheme



Attacking HORS

Example

- As on the previous slide, suppose that $\sigma(m_1) = \{13, 21, 25, 27\}$.
- Suppose that some additional messages are signed, where the selection function yields

$$\sigma(m_2) = \{5, 6, 13, 24\}$$
$$\sigma(m_3) = \{2, 21, 26, 30\}$$
$$\sigma(m_4) = \{5, 8, 18, 27\}$$
$$\sigma(m_5) = \{4, 9, 26, 29\}.$$

- Now suppose that

$$\sigma(m_6) = \{2, 13, 18, 29\}.$$

- Given signatures for $m_1, \ldots, m_5$, the signature for m_6 can be forged because the private keys $y_2, y_{13}, y_{18}, y_{29}$ have already been used.

Hash Chains

- The *Winternitz Signature Scheme* uses hash chains to allow multiple bits to be signed simultaneously.
- A hash chain looks like the following structure:

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

  where each arrow represents an application of the one-way function f.
- Thus, $y^j = f^j(y^0)$ for $1 \leq j \leq 7$, and $z = f^8(y^0)$, where f^j denotes j applications of the function f.
- In general, the hash chain would consist of $2^w + 1$ values, namely, $y^0, \ldots, y^{2^w-1}$ and z, where w is a specified parameter of the scheme.
- The example corresponds to $w = 3$.
- For a k-bit message, we would construct $\ell = k/w$ hash chains (we assume that k is a multiple of w, for convenience).
- Denote the initial values in these hash chains by $y_1^0, y_2^0, \ldots, y_\ell^0$. These initial values comprise the private key.
- The values $z_1, , \ldots, z_\ell$ comprise the public key.

An Insecure Scheme

- An ℓw-bit message can be parsed as $(x_1, \ldots, x_\ell)$, where each x_i is a binary w-tuple.
- We can view each x_i as an integer between 0 and $2^w - 1$ (inclusive).
- Suppose we release the values $a_i = y_i^{x_i} = f^{x_i}(y_i)$ for $i = 1, \ldots, \ell$ as a signature.
- The signer does not need to store the entire hash chains; they can compute the a_i's, as needed, from the initial values that form the private key.
- To verify a given a_i, simply check that $f^{2^w - x_i}(a_i) = z_i$.
- As we mentioned on the previous slide, the z_i's are the public key.

Example

Suppose $k = 9$, $\ell = 3$ and $w = 3$. There are three hash chains:

$$y_1^0 \to y_1^1 \to y_1^2 \to y_1^3 \to y_1^4 \to y_1^5 \to y_1^6 \to y_1^7 \to z_1$$
$$y_2^0 \to y_2^1 \to y_2^2 \to y_2^3 \to y_2^4 \to y_2^5 \to y_2^6 \to y_2^7 \to z_2$$
$$y_3^0 \to y_3^1 \to y_3^2 \to y_3^3 \to y_3^4 \to y_3^5 \to y_3^6 \to y_3^7 \to z_3.$$

Suppose Alice wants to sign the message 011101001. We have $x_1 = 011 = 3$, $x_2 = 101 = 5$ and $x_3 = 001 = 1$. So the signature consists of the values $a_1 = y_1^3$, $a_2 = y_2^5$, and $a_3 = y_3^1$.

$$y_1^0 \to y_1^1 \to y_1^2 \to \boxed{y_1^3} \to y_1^4 \to y_1^5 \to y_1^6 \to y_1^7 \to z_1$$
$$y_2^0 \to y_2^1 \to y_2^2 \to y_2^3 \to y_2^4 \to \boxed{y_2^5} \to y_2^6 \to y_2^7 \to z_2$$
$$y_3^0 \to \boxed{y_3^1} \to y_3^2 \to y_3^3 \to y_3^4 \to y_3^5 \to y_3^6 \to y_3^7 \to z_3.$$

Verification of the signature requires checking that

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

Security Issue

- Here is the problem: the released values are just elements in the three hash chains, and once an element in a hash chain is known, anyone can compute any later values in the hash chains, as desired.
- As an example, an adversary could compute

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

- Therefore, the adversary can now create the signature (y_1^5, y_2^6, y_3^4) for the "new" message 101110100:

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

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

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

Fixing the Problem

- To fix the problem, include a checksum in the message, and sign the checksum.
- The checksum is defined to be

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

In the previous example, we would have

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

- In binary, we have $C = 1100$. We sign three bits at a time, so we pad C on the left with two zeroes, and then break C into two chunks of three bits: $x_4 = 001 = 1$ and $x_5 = 100 = 4$.
- We need two additional hash chains, having public keys z_4 and z_5, to sign x_4 and x_5.

Fixing the Problem (cont.)

- The values $a_4 = y_4^1 = f(y_4)$ and $a_5 = y_5^4 = f^4(y_5)$ are included as part of the signature:

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

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

- The entire signature on the message (x_1, x_2, x_3) is $(a_1, a_2, a_3, a_4, a_5)$.
- To verify this signature, the following steps are performed:
  1. Verify that (a_1, a_2, a_3) is the correct signature for (x_1, x_2, x_3).
  2. Compute the checksum and create (x_4, x_5).
  3. Verify that (a_4, a_5) is the correct signature for (x_4, x_5).

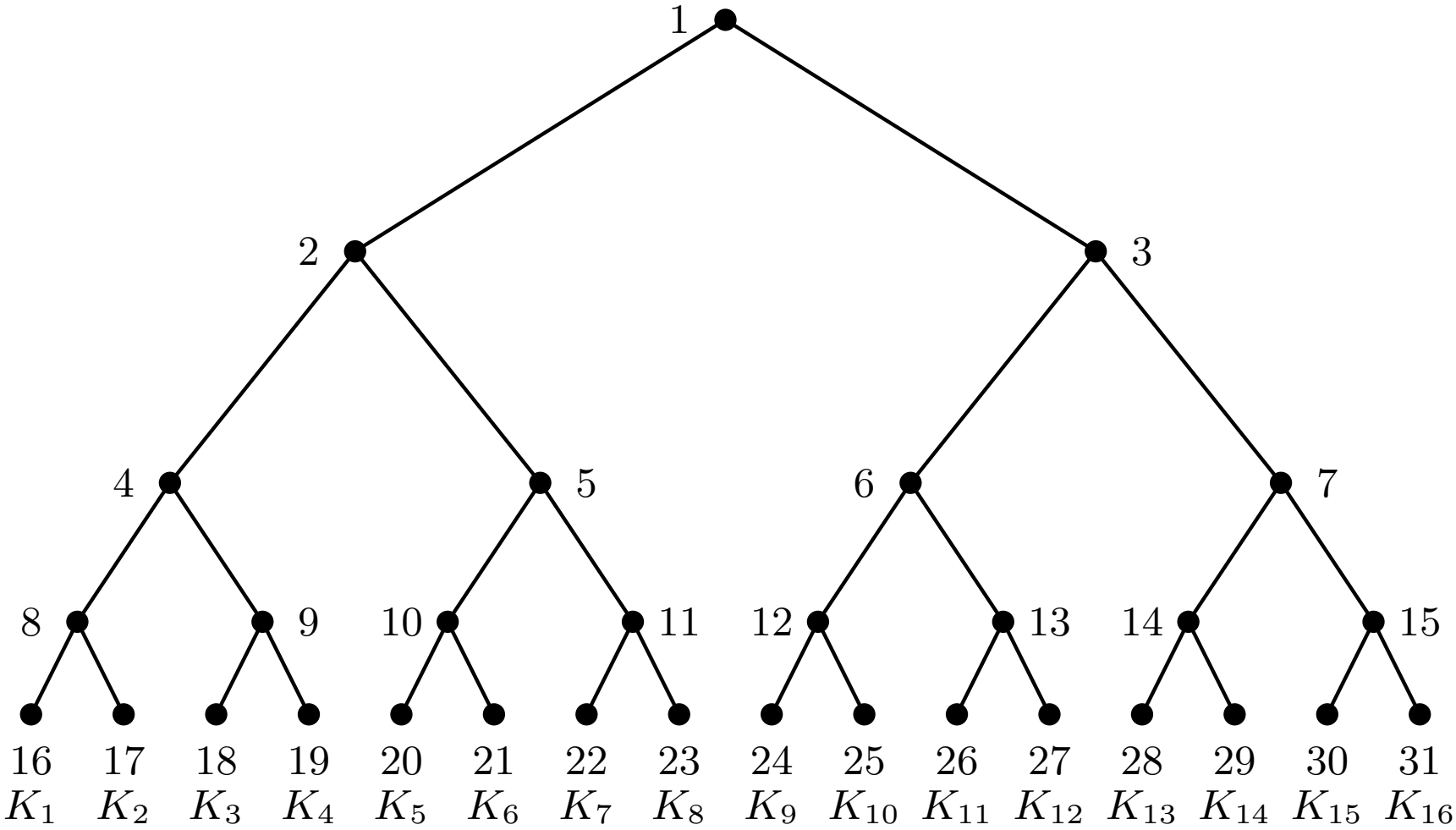
Introduction to Merkle Trees

- In 1979, Merkle invented a useful method of extending a one-time scheme so it could be used for a large (but fixed) number of signatures, without increasing the size of the public key.
- The one-time scheme that is used to sign messages is not important.
- The basic idea is to create a binary tree (which is now called a Merkle tree) by hashing combinations of various public keys (i.e., verification keys) of one-time signature schemes.
- The Merkle tree will be used only to authenticate public keys; it is not used to create signatures in the component one-time signature schemes.
- Conceptually, a Merkle tree is perhaps similar to a certificate chain.
- Let d be a prespecified positive integer, and suppose we have 2^d instances of a one-time signature scheme, with public verification keys denoted by $K_1, \ldots, K_{2^d}$, respectively.
- It is then possible to sign a series of 2^d messages, where the signature on the ith message will be verified using K_i, for $1 \leq i \leq 2^d$.

A Merkle Tree

- The Merkle tree is a complete binary tree, say $\mathcal{T}$, of depth d.
- The nodes of $\mathcal{T}$ are labelled so they satisfy the following properties:
  1. For $0 \leq \ell \leq d$, the 2^ℓ nodes at depth ℓ are labelled (in order) $2^\ell, 2^\ell + 1, \ldots, 2^{\ell+1} - 1$.
  2. For $j \neq 1$, the parent of node j is node $\lfloor \frac{j}{2} \rfloor$.
  3. The left child of node j is node $2j$ and the right child of node j is node $2j + 1$, assuming that one or both of these children exist.
  4. For $j \neq 1$, the sibling of node j is node $j + 1$, if j is even; or node $j - 1$, if j is odd.
- The next slide depicts a Merkle tree of depth $d = 4$.
- The keys $K_1, \ldots, K_{16}$ are associated with the leaf nodes $16, \ldots, 31$, respectively.
- So key K_i is associated with leaf node $i + 15$, for $1 \leq i \leq 16$.

A Merkle Tree of Depth 4 with 16 Leaf Nodes



Hashing Public Keys

- Let h be a secure cryptographic hash function.
- Each node j in $\mathcal{T}$ is assigned a value $V(j)$, according to the following rules.
  1. For $2^d \leq j \leq 2^{d+1} - 1$, let $V(j) = h(K_{j-2^d+1})$.
  2. For $1 \leq j \leq 2^d - 1$, let $V(j) = h(V(2j) \parallel V(2j+1))$.
- All the values $V(j)$ are strings of a fixed length, namely, the length of a message digest for the hash function h (e.g., 224 bits).
- The values stored in the 2^d leaf nodes are obtained by hashing the 2^d public keys.
- The value stored in a nonleaf node is computed by hashing the concatenation of the values stored in its two children.
- The value stored in the root node, which is $V(1)$, is the public key K for the scheme.

Signing and Verifying a Message

- We now discuss how to create a signature for the ith message, say m_i.
- First, the ith private (signing) key is used to create a signature s_i for the message m_i.
- This signature can be verified using the public key K_i, which must also be supplied as part of the signature.
- In addition, this public key K_i must be authenticated, which involves Merkle tree $\mathcal{T}$.
- This is done by providing enough information for the verifier to be able to recompute the value in the root, $V(1)$, and compare it to the stored value K.
- This necessary information consists of $V(i + 2^d - 1) = h(K_i)$, along with the values of the siblings of all the nodes in the path in $\mathcal{T}$ from node $i + 2^d - 1$ to the root node (i.e., node 1).

Example

Suppose $d = 4$ and suppose we want to sign message m_{11}. The relevant path contains nodes 26, 13, 6, 3, and 1. The siblings of the nodes on this path are nodes 27, 12, 7, and 2, so $V(27)$, $V(12)$, $V(7)$, and $V(2)$ are supplied as part of the signature. The entire signature consists of the list

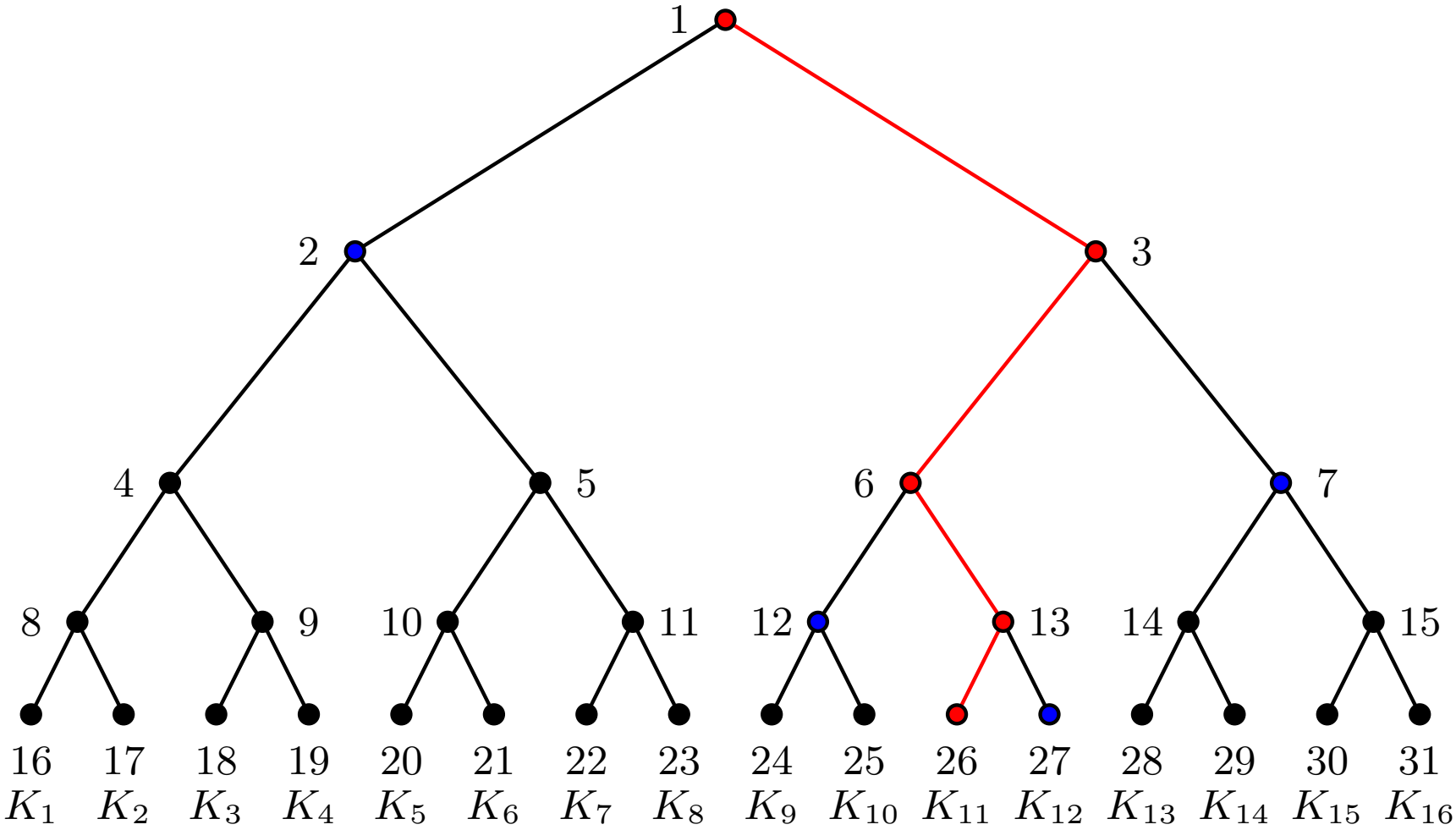
$$K_{11}, s_{11}, V(27), V(12), V(7), V(2).$$

The public key K_{11} would be authenticated as follows:

1. compute $V(26) = h(K_{11})$
2. compute $V(13) = h(V(26) \| V(27))$
3. compute $V(6) = h(V(12) \| V(13))$
4. compute $V(3) = h(V(6) \| V(7))$
5. compute $V(1) = h(V(2) \| V(3))$
6. verify that $V(1) = K$ (K is the root public key).

Finally, the signature s_{11} on m_{11} is verified using K_{11}.

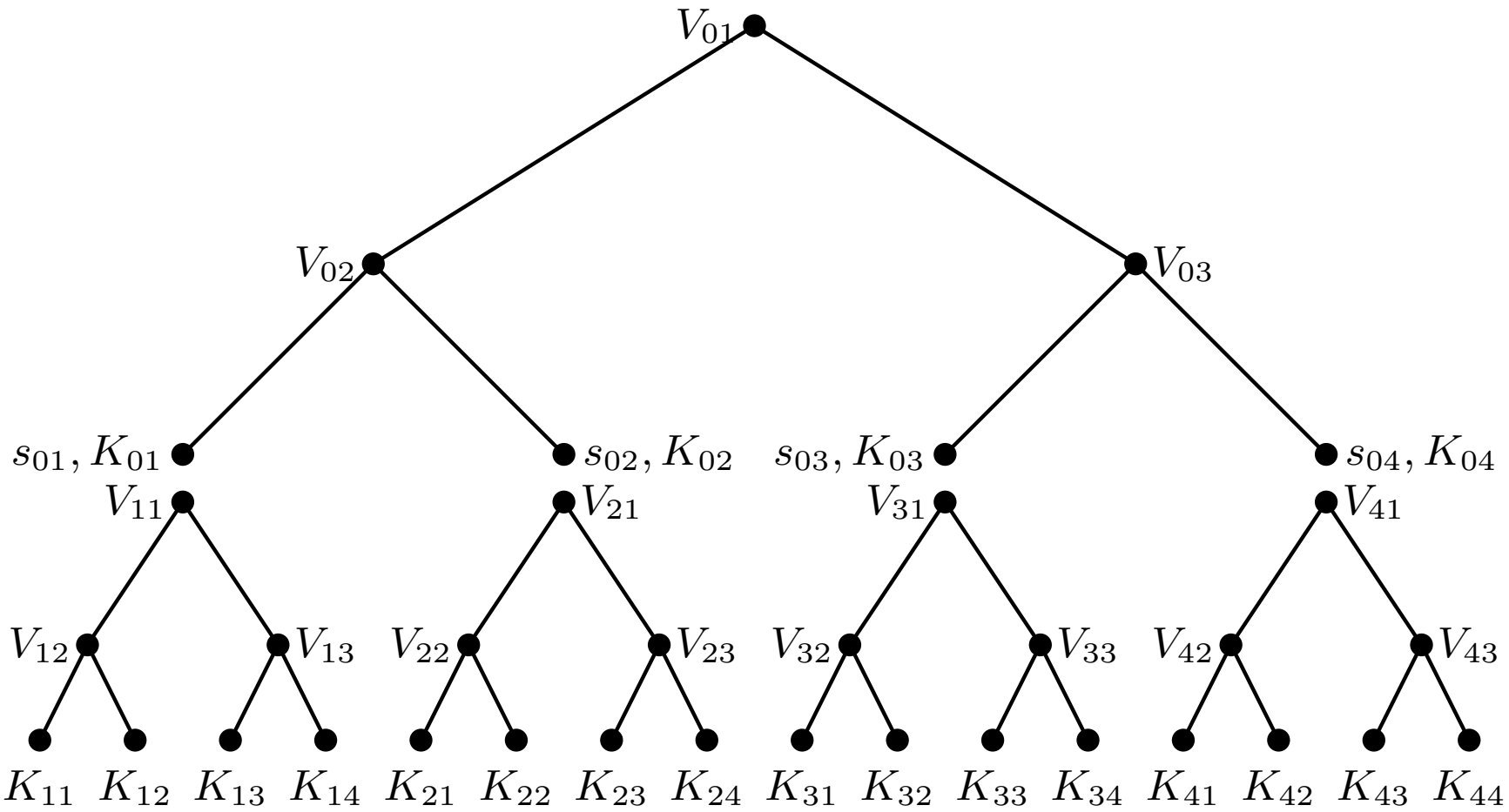
Authenticating the key K_{11}



Multilevel Merkle Trees

- Instead of using one large Merkle tree, it is more efficient to use multi-level Merkle trees.
- We describe a two-level Merkle tree.
- The leaf nodes of the top-level Merkle tree are use to sign the root nodes of the second-level Merkle trees.
- The leaf nodes of the second-level Merkle trees are use to sign messages.
- If the trees are of depth d, we have one top-level tree and 2^d second-level trees.
- The total number of leaf nodes is 2^{2d}.
- The keys in any one of these trees are independent of the keys in any other tree.

A Two-level Merkle Tree



Example

Let s be the signature on a message m that will be verified with key K_{33}. The entire signature consists of the list

$$K_{33}, s, K_{34}, V_{32}, s_{03}, K_{03}, K_{04}, V_{02}.$$

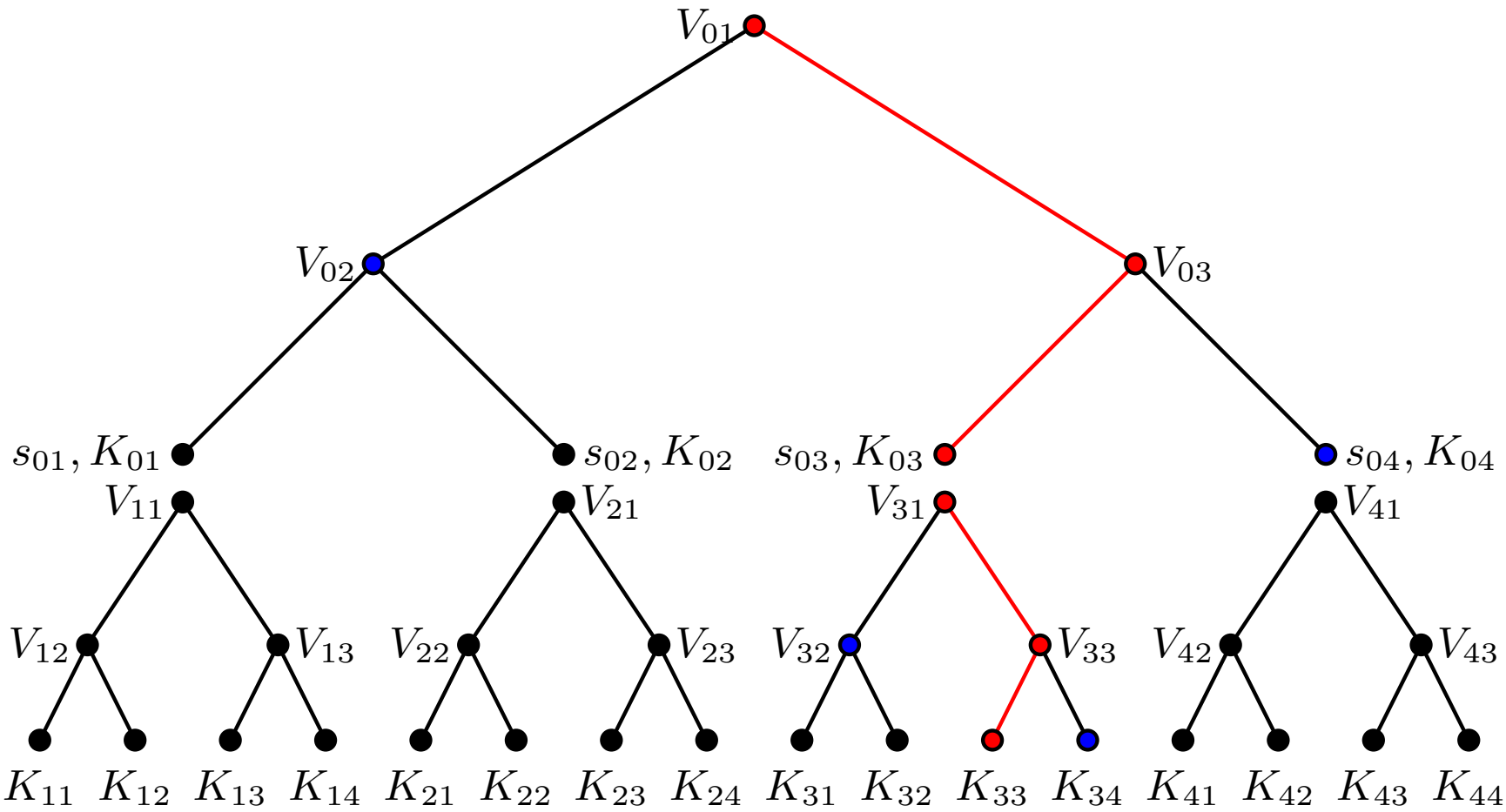
Note that s_{04} is not part of the signature, since we are not authenticating V_{41}.

The key K_{33} would be authenticated as follows:

1. compute $V_{33} = h(K_{33} \parallel K_{34})$
2. compute $V_{31} = h(V_{32} \parallel V_{33})$
3. use K_{03} to verify that s_{03} is a valid signature on V_{31}
4. compute $V_{03} = h(K_{03} \parallel K_{04})$
5. compute $V_{01} = h(V_{02} \parallel V_{03})$
6. verify that $V_{01} = K$ (K is the root public key).

Finally, the signature s on message m is verified using K_{33}.

Verifying a Signature Using a Two-level Merkle Tree



Stateful vs Stateless Signatures

- The Merkle tree-based schemes that we have described are stateful.
- A sequence of messages, say $m_1, m_2, \ldots$, is signed, and the keys used for creating and verifying for the ith signature depends on the index i.
- The signer would increment the index i every time a message is signed, which guarantees that a public-private key pair is not re-used.
- NIST approved two standards for stateful hash-based signatures in 2018; see NIST SP 800-208, Recommendation for Stateful Hash-Based Signature Schemes.
- These standards are based on the *Leighton-Micali Signature Scheme* and the *eXtended Merkle Signature Scheme*.

Stateful vs Stateless Signatures (cont.)

- The current standardization of post-quantum signature schemes is seeking stateless schemes.
- Stateless hash-based signature schemes can be constructed using a multilevel Merkle tree that supports an enormous number of keys.
- A leaf node of a bottom-level tree can be picked randomly every time a new message is signed and we can be confident that keys are not re-used.
- Furthermore, in order to mitigate the consequences of accidentally re-using keys, we can replace the one-time signature schemes employed at the leaf nodes of the bottom level trees—these are the schemes that actually sign the messages—by few-times signature schemes.
- *SPHINCS*$^+$ incorporates many of the techniques we have discussed; it was standardized as FIPS 205.

3 Code-based cryptography

- Linear codes
- McEliece cryptosystem
- Niederreiter cryptosystem
- Information-set decoding
- BIKE
- Hamming Quasicyclic (HQC)

Definitions

- The purpose of an error-correcting code is to permit the correction of random errors that occur in the transmission of a message over a noisy channel.
- We study binary linear codes as a solution to this problem.
- Let k, n be positive integers, $k \leq n$.
- A $[n, k]$ code, $\mathbf{C}$, is a k-dimensional subspace of $(\mathbb{Z}_2)^n$, the vector space of all binary n-tuples.
- The code is binary because the alphabet is $\{0, 1\}$.
- The code is linear because it is a subspace of a vector space.
- A generator matrix for an $[n, k]$ code, $\mathbf{C}$, is a $k \times n$ binary matrix whose rows form a basis for $\mathbf{C}$.
- Suppose $\mathbf{x}$ is a binary k-tuple (i.e., a message).
- We encode $\mathbf{x}$ as the n-tuple $\mathbf{y} = \mathbf{x}G$ and we transmit the codeword $\mathbf{y}$ through the communication channel to a receiver.

Definitions (cont.)

- Some random errors may be introduced while $\mathbf{y}$ is being transmitted through the communication channel.
- The vector received by Bob is $\mathbf{r} = \mathbf{y} + \mathbf{e}$, where $\mathbf{e}$ is the error vector.
- We refer to $\mathbf{r}$ as the received vector.
- The code should have sufficient redundancy to allow $\mathbf{y}$ to be reconstructed from $\mathbf{r}$; this is the decoding problem.
- Finally, given $\mathbf{y}$, it is straightforward to determine $\mathbf{x}$.

Example

- The matrix

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

  is a generating matrix for a $[6, 3]$ code.

- The code $\mathbf{C}$ contains eight codewords:

$$\begin{array}{ll} (0,0,0,0,0,0) & (1,0,0,0,1,1) \\ (0,1,1,0,1,0) & (0,0,1,1,0,1) \\ (1,1,1,0,0,1) & (1,0,1,1,1,0) \\ (0,1,0,1,1,1) & (1,1,0,1,0,0) \end{array}$$

- The message $\mathbf{x} = (1,1,0)$ is encoded to the codeword $\mathbf{y} = (1,1,1,0,0,1)$.

Distance of a Code

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

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

  i.e., the number of co-ordinates in which $\mathbf{x}$ and $\mathbf{y}$ differ.
- Let $\mathbf{C}$ be an $[n, k]$ code. Define the distance of $\mathbf{C}$ to be the quantity

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

- An $[n, k, d]$ code is an $[n, k]$ code, say $\mathbf{C}$, in which $dist(\mathbf{C}) \geq d$.
- The weight of a $\mathbf{x} \in (\mathbb{Z}_2)^n$ is the number of nonzero co-ordinates in $\mathbf{x}$, namely $|\{i : x_i \neq 0\}|$.
- It is easy to prove that $dist(\mathbf{C})$ is equal to the minimum weight of a nonzero codeword.

We recall the example from slide #58.

Example

- The matrix

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

  is a generating matrix for a $[6, 3]$ code.

- The code $\mathbf{C}$ contains eight codewords:

$$\begin{array}{ll} (0,0,0,0,0,0) & (1,0,0,0,1,1) \\ (0,1,1,0,1,0) & (0,0,1,1,0,1) \\ (1,1,1,0,0,1) & (1,0,1,1,1,0) \\ (0,1,0,1,1,1) & (1,1,0,1,0,0) \end{array}$$

- The codewords have weights 0, 3, 3, 3, 4, 4, 4, and 3, so $\mathit{dist}(\mathbf{C}) = 3$.

Parity-check Matrices

- Two vectors $\mathbf{x}, \mathbf{y} \in (\mathbb{Z}_2)^n$, say $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, are orthogonal if
$$\sum_{i=1}^{n} x_i y_i \equiv 0 \pmod 2.$$
- The orthogonal complement of a linear $[n, k, d]$ code, $\mathbf{C}$, consists of all the vectors that are orthogonal to all the vectors in $\mathbf{C}$.
- This set of vectors is denoted by $\mathbf{C}^{\perp}$; it is the dual code to $\mathbf{C}$.
- $\mathbf{C}^{\perp}$ is a linear code having dimension $n - k$.
- A parity-check matrix for a linear $[n, k, d]$ code $\mathbf{C}$ having generating matrix G is a generating matrix H for $\mathbf{C}^{\perp}$.
- This matrix H is an $(n - k)$ by n matrix.
- The rows of H are linearly independent vectors, and GH^T is a k by $n - k$ matrix of zeroes.
- It can be proven that $dist(\mathbf{C})$ is equal to the minimum number of columns of H that sum to the zero vector.

Decoding

- Suppose an $[n, k, d]$ code is used to transmit messages from Alice to Bob.
- As we mentioned earlier, Bob receives the n-tuple $\mathbf{r}$, which may not be the same as the codeword $\mathbf{y}$ transmitted by Alice.
- He will decode $\mathbf{r}$ using the strategy of nearest neighbour decoding.
- Bob finds a codeword $\mathbf{y}' \neq \mathbf{r}$ that has minimum distance to $\mathbf{r}$.
- Such a codeword $\mathbf{y}'$ will be called a nearest neighbour to $\mathbf{r}$ and it will be denoted as by $nn(\mathbf{r})$ (note that it is possible that there might be more than one nearest neighbour).
- Then Bob decodes $\mathbf{r}$ to $nn(\mathbf{r})$.
- If at most $(d-1)/2$ errors occurred during transmission, then nearest neighbour decoding corrects all the errors: any received vector $\mathbf{r}$ will have a unique nearest neighbour, and $nn(\mathbf{r}) = \mathbf{y}$.
- The main problem with nearest neighbour decoding is that it is very inefficient.

Syndrome Decoding

- Suppose $\mathbf{C}$ is a linear $[n, k]$ code having parity-check matrix H.
- Given a vector $\mathbf{r} \in (\mathbb{Z}_2)^n$, we define the syndrome of $\mathbf{r}$ to be $H\mathbf{r}^T$.
- A syndrome is a column vector with $n - k$ components. It can be shown that $\mathbf{r} \in (\mathbb{Z}_2)^n$ is a codeword if and only if

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

- Further, if $\mathbf{y} \in \mathbf{C}$, $\mathbf{e} \in (\mathbb{Z}_2)^n$ and we define $\mathbf{r} = \mathbf{y} + \mathbf{e}$, then $H\mathbf{r}^T = H\mathbf{e}^T$.
- Thus, the syndrome is equal to the sum of the columns of H corresponding to the errors.
- This observation is the basis of certain decoding algorithms.

Example

The matrix

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

is a generating matrix for a linear $[7, 4, 3]$ code, known as a Hamming code. The matrix

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

is a parity-check matrix.

Syndrome Decoding (cont.)

- The Hamming code has distance three, so it will correct one error.
- If there are no errors, then the syndrome will be a column vector of 0's.
- If there is one error, then the syndrome will be one of the columns of H.
- If the error occurs in the ith position of $\mathbf{y}$, then the syndrome will be the ith column of H.

Example

- $\mathbf{x} = (1, 0, 1, 1)$ will be encoded as $\mathbf{y} = (1, 0, 1, 1, 0, 1, 0)$.
- Suppose $\mathbf{r} = (1, 1, 1, 1, 0, 1, 0)$ is received.
- The syndrome is $\mathbf{s} = H\mathbf{r}^T = (1, 0, 1)^T$.
- This is the second column of H, so we would decode $\mathbf{r}$ to $\mathbf{r} + (0, 1, 0, 0, 0, 0, 0) = \mathbf{y}$.

Goppa Codes and the McEliece Cryptosystem

- Goppa codes were invented in 1970.
- The Goppa code $\Gamma(L, g)$ is defined by a degree t polynomial $g(z) \in \mathbb{F}_{2^m}[z]$ and a set L of n elements of $\mathbb{F}_{2^m}$. We list some properties of the code $\Gamma(L, g)$.
- $\Gamma(L, g)$ is a binary linear code.
- The dimension k of $\Gamma(L, g)$ satisfies the inequality $k \geq n - mt$.
- The distance d of $\Gamma(L, g)$ satisfies the inequality $d \geq 2t + 1$; hence t or fewer errors can be corrected.
- Goppa codes can be efficiently encoded and decoded.
- McEliece proposed his public-key cryptosystem in 1978.
- The strategy is to start with a Goppa code and then disguise it so the underlying structure is obscured.
- An attacker is faced with the problem of decoding what appears to be a random linear code, which is an NP-complete problem.

McEliece Cryptosystem (Key Generation)

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

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

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

The public generating matrix is used to encrypt. The private key is used to permit decryption (in step 2) using the original Goppa code.

McEliece Cryptosystem (Encryption and Decryption)

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

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

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

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

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

The *Niederreiter Cryptosystem* (from 1986) is similar to the *McEliece Cryptosystem*, except it uses the parity check matrix for encryption.

Niederreiter Cryptosystem (Key Generation)

- Let H be a parity-check matrix for a linear $[n, k, d]$ Goppa code $\mathbf{C}$, where $n = 2^m$, $d = 2t + 1$, and $k = n - mt$.
- Let S be an $(n - k) \times (n - k)$ matrix that is invertible over $\mathbb{Z}_2$, let P be an $n \times n$ permutation matrix, and let $H' = SHP$.
- Let $\mathcal{P}$ consist of all binary n-tuples having weight t, let $\mathcal{C}$ consist of all binary column vectors of length $n - k$, and let

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

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

Niederreiter Cryptosystem (Encryption and Decryption)

For a public key H', a plaintext $\mathbf{x} \in \mathcal{P}$ is encrypted by computing

$$\mathbf{y} = H'\mathbf{x}^T.$$

Note that $\mathbf{y}$ is a column vector of length $n - k$.

A ciphertext $\mathbf{y}$ is decrypted by means of the following operations:

1. Compute $S^{-1}\mathbf{y}$. Observe that $S^{-1}\mathbf{y} = HP\mathbf{x}^T$, which is the syndrome of the vector $P\mathbf{x}^T$ that has weight t.
2. Decode $S^{-1}\mathbf{y}$, obtaining $P\mathbf{x}^T$.
3. Compute $\mathbf{x}^T = P^{-1}(P\mathbf{x}^T)$.

Discussion

- The *Niederreiter Cryptosystem* has similar security to the *McEliece Cryptosystem*, but encryption is more efficient.
- Plaintexts are actually correctible error patterns; they are binary vectors of a specified weight t.
- In practice, we would probably want to encrypt an arbitrary binary vector of a given length.
- Let $m = \lfloor \log_2 \binom{n-k}{t} \rfloor$.
- It is relatively straightforward to find an injective mapping f from the set of all binary vectors of length m to the set of all binary $(n-k)$-tuples having weight t.
- Then we can start with a "plaintext" $\mathbf{z} \in (\mathbb{Z}_2)^m$ and use the *Niederreiter Cryptosystem* to encrypt $\mathbf{x} = f(\mathbf{z})$.
- After we decrypt $\mathbf{y}$, obtaining $\mathbf{x}$, we compute $\mathbf{z} = f^{-1}(\mathbf{x})$.
- The NIST submission *Classic McEliece* (a Round 4 finalist) is actually a *Niederreiter Cryptosystem*.

Generic Attacks

- The most effective attacks against the *McEliece Cryptosystem* and the *Niederreiter Cryptosystem* are generic attacks that do not make use of the fact that the code is a Goppa code.
- The general syndrome decoding problem for a linear code can be rephrased as follows: given a syndrome $\mathbf{s}$, find a subset of at most t columns of H whose sum is $\mathbf{s}$.
- An inefficient brute-force approach would involve computing this sum for all t-subsets of the n columns of H, of which there are $\binom{n}{t}$.
- *Information-set decoding*, introduced in 1962 by Prange, is the basis of several improved algorithms.

Prange Information-set Decoding

- We describe information-set decoding, assuming that we have a generator matrix G of a $[n, k, d = 2t + 1]$-binary code.
- Let $J \subseteq \{1, \ldots, n\}$, $|J| = k$.
- Let G' denote the restriction of G to the columns labelled by J. If G' is invertible, then J is called an information set.
- As usual, let $\mathbf{r} = \mathbf{y} + \mathbf{e}$, where $wt(\mathbf{e}) \leq t$.
- Suppose that no errors occurred in the co-ordinates labelled by J, so $\mathbf{r}' = \mathbf{y}'$ (again we are restricting to the co-ordinates indexed by J).
- Given $\mathbf{y}'$, it is easy to generate the codeword $\mathbf{y}$.
- If $\mathbf{r}' = \mathbf{y}'$, then the codeword generated from $\mathbf{r}'$ must be $\mathbf{y}$.
- Observe that $\mathbf{y}' = \mathbf{x}G'$, so $\mathbf{x} = \mathbf{y}'(G')^{-1}$.
- Then

$$\mathbf{y} = \mathbf{x}G = \mathbf{y}'(G')^{-1}G = \mathbf{r}'(G')^{-1}G.$$

Prange Information-set Decoding (cont.)

This suggests the following randomized algorithm.

1. Input: Given generator matrix G for a $[n, k, d = 2t + 1]$-binary code and a received vector $\mathbf{r}$ in which at most t errors have occurred.
2. Randomly choose $J \subseteq \{1, \ldots, n\}$, $|J| = k$.
3. If J is an information set (i.e., if G' is invertible), then proceed to step 4. Otherwise, return to step 2.
4. Compute the codeword $\mathbf{y}$ such that $\mathbf{y}' = \mathbf{r}'$:

$$\mathbf{y} = \mathbf{r}'(G')^{-1}G.$$

5. Check if $dist(\mathbf{y}, \mathbf{r}) \leq t$. If so, decoding has succeeded. Otherwise, return to step 2.

Ignoring the possibility that J is not an information set, the success probability of this algorithm is $\binom{n-t}{k} / \binom{n}{k}$.

Example

We use the following generator matrix from slide #58:

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

Suppose that $\mathbf{r} = (1, 1, 0, 1, 0, 1)$. We randomly choose $J = \{1, 2, 3\}$. Then

$$G' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad (G')^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

Since $\mathbf{r}' = (1, 1, 0)$, we obtain

$$\mathbf{y} = (1, 1, 0) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} G = (1, 1, 1)G = (1, 1, 0, 1, 0, 0).$$

Since $dist(\mathbf{C}) = 3$ and $dist(\mathbf{y}, \mathbf{r}) = 1$, we decode $\mathbf{r}$ to $\mathbf{y}$.

Block-circulant Matrices and BIKE

- *BIKE* (*Bit-flipping key encapsulation*) was a Round 4 NIST finalist. We describe some of its main features now.
- A circulant matrix is a square matrix in which row is a cyclic shift of the previous row by one position to the right.
- A circulant r by r binary matrix corresponds to an ideal in the quotient ring $\mathcal{R} = \mathbb{Z}_2[x]/(x^r - 1)$.
- For a polynomial $a \in \mathcal{R}$, let $wt(a)$ denote the number of nonzero coefficients in a.
- In *BIKE*, r is prime and 2 is a primitive element in $\mathbb{Z}_r$.
- As a consequence of these properties of r, it can be proven that the irreducible factors of $x^r - 1$ in $\mathbb{Z}_2[x]$ are $x - 1$ and $x^{r-1} + \cdots + 1$.
- Further, $a \in \mathcal{R}$ is invertible if and only if $wt(a)$ is odd and $wt(a) \neq r$.

Block-circulant Matrices and BIKE (cont.)

- A block circulant matrix is made up of blocks, each of which is a circulant (sub)matrix.
- *BIKE* has a secret parity-check matrix which is a block circulant matrix with two blocks.
- The parity-check matrix is an MDPC code (medium-density parity-check code).
- These are extensions of the classic LDPC codes (low-density parity-check codes).
- Roughly speaking, an LDPC code typically has constant row weight, whereas an MDPC code has a somewhat higher row weight.
- Denote $n = 2r$.
- Let $w \approx \sqrt{n}$ be chosen such that $w/2$ is odd (w is the row weight).
- Let $t \approx \sqrt{n}$ (t is the error weight).

Block-circulant Matrices and BIKE (cont.)

- Recall that $\mathcal{R} = \mathbb{Z}_2[x]/(x^r - 1)$.
- Let $h_0, h_1 \in \mathcal{R}$ be chosen such that $wt(h_0) = wt(h_1) = w/2$.
- Let $h = h_1(h_0)^{-1}$ (h is part of the public key, to be described a bit later).
- The computation of h is done in $\mathcal{R}$ using the *Extended Euclidean algorithm*.
- The private key is the pair (h_0, h_1).
- Note that this private key generates an r by n block circulant matrix H consisting of two r by r blocks.
- The matrix H is the parity check matrix of a binary linear code.
- H is completely determined by its first row due to the block circulant structure.
- Every row of H has weight w.

Example

- Suppose $r = 11$, so $n = 22$, and let $w = 6$.
- Let $h_0(x) = x^9 + x^5 + x^4$ and $h_1(x) = x^{10} + x + 1$.
- The (private) parity-check matrix H is

$$\left(\begin{array}{ccccccccccc|ccccccccccc} 0&0&0&0&1&1&0&0&0&1&0 & 1&1&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&1&1&0&0&0&1 & 1&1&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&1&1&0&0&0 & 0&1&1&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&1&0&0 & 0&0&1&1&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&1&1&0 & 0&0&0&1&1&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&1&1 & 0&0&0&0&1&1&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0&0&0&1 & 0&0&0&0&0&1&1&1&0&0&0 \\ 1&1&0&0&0&1&0&0&0&0&0 & 0&0&0&0&0&0&1&1&1&0&0 \\ 0&1&1&0&0&0&1&0&0&0&0 & 0&0&0&0&0&0&0&1&1&1&0 \\ 0&0&1&1&0&0&0&1&0&0&0 & 0&0&0&0&0&0&0&0&1&1&1 \\ 0&0&0&1&1&0&0&0&1&0&0 & 1&0&0&0&0&0&0&0&0&1&1 \end{array}\right)$$

- The first circulant block is generated by $h_0(x)$ and the second circulant block is generated by $h_1(x)$.
- The two polynomials are written as binary vectors, where the degree of the terms increases from left to right.

BIKE

- The public key consists of the polynomial h and the error weight t.
- A plaintext is a pair (e_0, e_1) where $wt(e_0) + wt(e_1) = t$.
- We can view a plaintext as an error vector of length n and weight t.
- The encryption of (e_0, e_1) is $s = e_0 + e_1 h$.
- A ciphertext is basically a syndrome computed using the public parity-check matrix H_{pub} generated from $(1, h)$.
- Decryption is done as follows:
  1. Compute $m = sh_0$. Since $hh_0 = h_1$, we have $m = e_0 h_0 + e_1 h_1$, and therefore m is the syndrome of (e_0, e_1) that is computed using the parity-check matrix H.
  2. Decrypt m using a bit-flipping algorithm.

Example

- Recall that $h_0(x) = x^9 + x^5 + x^4$ and $h_1(x) = x^{10} + x + 1$.
- We have

$$\begin{aligned} h_0^{-1}(x) &= x^9 + x^8 + x^7 + x^6 + x^5 + x^2 + 1 \\ h(x) &= x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + 1. \end{aligned}$$

- The (public) parity-check matrix H_{pub} is

$$\left(\begin{array}{cccccccccccc|cccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}\right)$$

- The first circulant block is generated by 1 and the second circulant block is generated by $h(x)$.

Example

- Suppose we take $t = 4$ and the plaintext consists of $e_0(x) = x^3$ and $e_1(x) = x^4 + x^7 + x^{10}$.
- The total weight of e_0 and e_1 is $t = 4$.
- The ciphertext is $s(x) = x^{10} + x^9 + x^8 + x^7 + x^5 + x^3$.
- This is equivalent to computing the matrix product $H_{\text{pub}}\mathbf{e}^T$, where

$$\mathbf{e} = (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0|1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0).$$

- Note that exponents in this coefficient vector are decreasing from left to right.
- The first step of decryption is to compute

$$m(x) = s(x)h_0(x) = x^{10} + x^9 + x^6 + x^5 + x^4 + x^3 + x + 1.$$

- m is the syndrome computed using the parity-check matrix H.
- In vector form, this syndrome is $(1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1)^T$.

Example

Suppose we sum the columns of H corresponding to the four nonzero monomials in e_0 and e_1:

$$\begin{array}{cccc} 0 & 1 & 0 & 0 \to 1 \\ 0 & 1 & 0 & 0 \to 1 \\ 1 & 0 & 1 & 0 \to 0 \\ 1 & 0 & 1 & 0 \to 0 \\ 0 & 0 & 1 & 0 \to 1 \\ 0 & 0 & 0 & 1 \to 1 \\ 0 & 0 & 0 & 1 \to 1 \\ 0 & 0 & 0 & 1 \to 1 \\ 0 & 0 & 0 & 0 \to 0 \\ 1 & 0 & 0 & 0 \to 1 \\ 0 & 1 & 0 & 0 \to 1 \end{array}$$

This yields the syndrome $(1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1)^T$.

Hamming Quasicyclic (HQC)

- The other Round 4 finalist is *HQC*, which is an abbreviation for *Hamming Quasicyclic*.
- *HQC* was in fact chosen by NIST in March 2025 to be standardized (see NIST Internal Report NIST IR 8545).
- The *HQC* cryptosystem incorporates two codes:
  1. An $[n, k, d]$-binary code $\mathcal{C}$ that has an efficient decoding algorithm. This code will have a public k by n generator matrix G.
  2. A random $[2n, n]$-binary quasicyclic code $\mathcal{H}$. This code has a public n by $2n$ parity check matrix H. The first n by n block is an n by n identity matrix and the second n by n block consists of rotations of an initial row $\mathbf{h}$. Thus
$$H = \left(\; I_n \;\middle|\; \mathrm{rot}(\mathbf{h}) \; \right).$$

Hamming Quasicyclic (HQC)

- The high-level strategy of *HQC* is to encrypt a plaintext using the generator matrix G to create a codeword in $\mathcal{C}$ and then add an error to it. The resulting vector $\mathbf{v}$ is part of the ciphertext. However, there are some differences from previous schemes we have discussed:
  1. The code $\mathcal{C}$ having generator matrix G is easy to decode, but G is not "disguised" in any way.
  2. The error vector added to the codeword in $\mathcal{C}$ has a large hamming weight, the consequence of which is that $\mathbf{v}$ cannot be decoded.
  3. There is a second component to the ciphertext, namely a vector $\mathbf{u}$, than enables $\mathbf{v}$ to be modified to a different vector that can be decoded. The way this modification is done is that part of the error vector is removed using knowledge of the private key together with $\mathbf{u}$. The vector $\mathbf{u}$ is a syndrome in the code $\mathcal{H}$, computed from the parity-check matrix H.

Time for a Coffee Break!

4 Lattices and lattice-based cryptography

- Introduction to lattices
- The LLL algorithm
- NTRU
- Learning with errors
- Regev cryptosystem
- Ring learning with errors
- Lyubashevsky, Peikert and Regev cryptosystem
- Kyber-PKE and Crystals-Dilithium

Vector Spaces

- A lattice is very similar to a vector space.
- A real vector space can be defined by starting with a basis, which is a set $\mathbf{B}$ of linearly independent vectors in $\mathbb{R}^n$ for some integer n.
- Vectors are just n-tuples in which each co-ordinate is an element of $\mathbb{R}$.
- The vector space generated or spanned by the given basis consists of all linear combinations of basis vectors.
- If there are r vectors in the basis $\mathbf{B}$, then we have an r-dimensional vector space.
- Restating this using mathematical notation, suppose that the basis is $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_r\}$. The vector space generated by $\mathbf{B}$ consists of all the vectors of the form

$$\alpha_1 \mathbf{b}_1 + \cdots + \alpha_r \mathbf{b}_r,$$

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

Lattices

- A lattice is very similar, except that the vectors in the lattice are integer linear combinations of basis vectors.
- That is, the lattice $\mathcal{L}(\mathbf{B})$ generated by the basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_r\}$ consists of all the vectors of the form

$$\alpha_1 \mathbf{b}_1 + \cdots + \alpha_r \mathbf{b}_r,$$

  where $\alpha_1, \ldots, \alpha_r$ are arbitrary integers.
- We are primarily interested in full rank lattices, i.e., those with $r = n$.
- For a vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$, we define the Euclidean norm of $\mathbf{v}$ to be

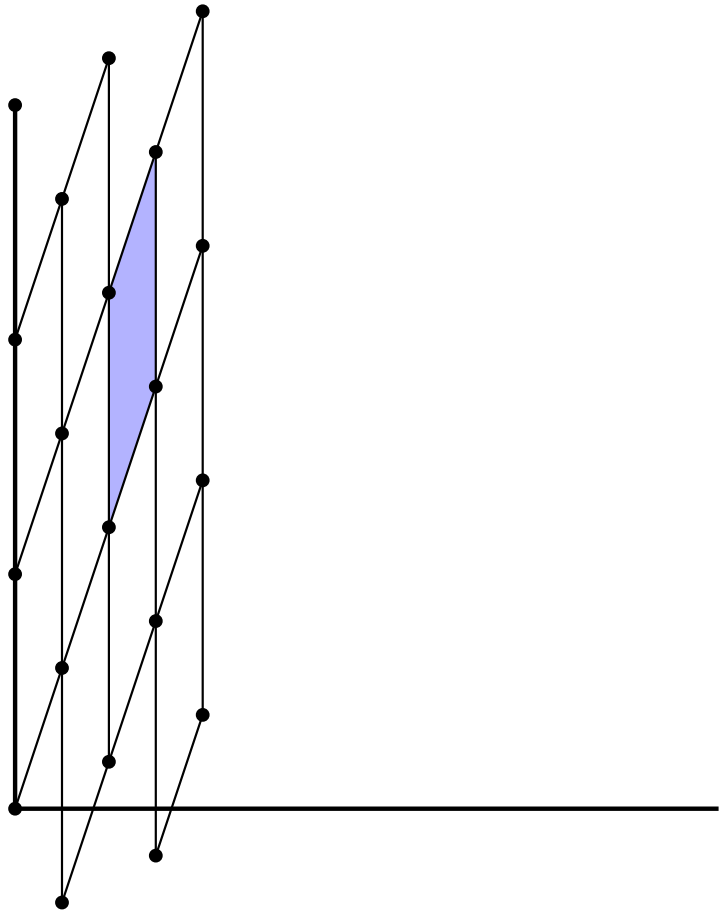
$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} {v_i}^2}.$$

- The infinity norm of $\mathbf{v}$ is

$$\|\mathbf{v}\|_\infty = \max\{|v_i| : 1 \leq i \leq n\}.$$

Example

We show part of the lattice defined by the basis $\{(0, 5), (1, 3)\}$. One fundamental parallelepiped is shaded.



Two Fundamental Problems

Shortest Vector

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

Closest Vector

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

There is an efficient Turing reduction from **Shortest Vector** to **Closest Vector**. Therefore **Shortest Vector** can be solved efficiently, assuming the existence of an oracle solving **Closest Vector**.

The LLL Algorithm

- Assume we have an n-dimensional lattice $\mathcal{L}$ in $\mathbb{R}^n$.
- The famous *LLL algorithm* (discovered in 1982) finds a so-called reduced basis for $\mathcal{L}$.
- Note that "LLL" is an abbreviation for *Lenstra-Lenstra-Lovasz*.
- The first vector in a reduced basis is relatively short, and if we are lucky, it might in fact be a shortest vector in $\mathcal{L}$.
- Various modifications and improvements of the *LLL algorithm* have been proposed.
- Many computer algebra systems have built-in implementations of lattice basis reduction algorithms.

Using the LLL Algorithm to Solve CVP

- Sometimes it is possible to use the *LLL algorithm* to solve the **Closest Vector** problem.
- Suppose we are given a basis $S = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ for a lattice $\mathcal{L}$ and a vector $\mathbf{w} \notin \mathcal{L}$.
- Let M be a small positive integer.
- Consider the $(n+1)$-dimensional lattice $\mathcal{L}'$ having basis

$$\{(\mathbf{v}_1, 0), \ldots, (\mathbf{v}_n, 0), (\mathbf{w}, M)\}.$$

- Then run the *LLL algorithm* on this basis of $\mathcal{L}'$.
- We could try various values of M, e.g., $M = 1, 2, \ldots$.
- It can be proven that, if M is close to the minimum distance of a vector in $\mathcal{L}'$ to $\mathbf{w}$, then the shortest vector in $\mathcal{L}'$ will yield the closest vector to $\mathbf{w}$.
- Specifically, $\mathbf{w} - \mathbf{r}_1 \in \mathcal{L}$ is close to $\mathbf{w}$ if the first row of the LLL-reduced matrix is $(\mathbf{r}_1, M)$.

Example

- Suppose $\mathcal{L}$ is the lattice whose basis vectors are the rows of the following matrix:

$$\begin{pmatrix} 10 & 7 & 4 & 6 & 8 \\ 3 & 11 & 13 & 2 & 20 \\ 5 & 9 & 9 & 3 & 14 \\ 6 & 8 & 2 & 19 & 11 \\ 4 & 10 & 9 & 17 & 6 \end{pmatrix}$$

  and suppose $\mathbf{w} = (38, 154, 198, 79, 170)$.

- If $M = 1$, then the basis for $\mathcal{L}'$ consists of the rows of the following matrix:

$$\begin{pmatrix} 10 & 7 & 4 & 6 & 8 & 0 \\ 3 & 11 & 13 & 2 & 20 & 0 \\ 5 & 9 & 9 & 3 & 14 & 0 \\ 6 & 8 & 2 & 19 & 11 & 0 \\ 4 & 10 & 9 & 17 & 6 & 0 \\ 38 & 154 & 198 & 79 & 170 & 1 \end{pmatrix}.$$

Example

- The LLL-reduced matrix is

$$\begin{pmatrix} 0 & 1 & -1 & 1 & 1 & 1 \\ -3 & 1 & 0 & -1 & 1 & 1 \\ -1 & 4 & 0 & -1 & -3 & 1 \\ -1 & 1 & 4 & 2 & 1 & 2 \\ -1 & -4 & 0 & 3 & -1 & 2 \\ -2 & 1 & -3 & 5 & 0 & -7 \end{pmatrix}.$$

- The last entry in the first row is $M = 1$.
- Hence

$$\mathbf{w} - (0, 1, -1, 1, 1) = (38, 153, 199, 78, 169) \in \mathcal{L}.$$

Introduction to NTRU

- *NTRU* is a public-key cryptosystem, due to Hoffstein, Pipher, and Silverman, that was introduced at the CRYPTO '96 rump session.
- It is a very fast cryptosystem that is easy to implement.
- It is also of interest because its security is related to certain lattice problems and thus it is considered to be a practical example of post-quantum cryptography.
- *NTRU* is defined in terms of three parameters, N, p and q, which are fixed integers.
- Computations are performed in the polynomial ring $\mathcal{R} = \mathbb{Z}[x]/(x^N - 1)$.
- Multiplication of two polynomials is easy in $\mathcal{R}$; it suffices to compute the product of two polynomials in $\mathbb{Z}[x]$ and then reduce all exponents modulo N.

Example

- Suppose $N = 3$ and we want to compute the product

$$(x^2 + 3x + 1)(2x^2 + x - 4)$$

in $\mathcal{R} = \mathbb{Z}[x]/(x^3 - 1)$.

- We compute as follows:

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

- We are using the facts that $x^3 = 1$ and $x^4 = x$.

NTRU Parameters

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

Definition

For an odd integer n and integers a and b, define

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

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

NTRU Public and Private Keys

Many versions of *NTRU* exist. We present a typical example. First, we describe the public and private keys.

1. $F(x)$ and $G(x)$ are secret polynomials chosen from $\mathcal{R}$. All coefficients of $F(x)$ and $G(x)$ will be in the set $\{-1, 0, 1\}$.
2. Define $f(x) = 1 + pF(x)$ and $g(x) = pG(x)$, and then discard $F(x)$ and $G(x)$.
3. Compute $f^{-1}(x)$ in the ring $\mathcal{R}$ mods q, and then compute $h(x) = f^{-1}(x)g(x)$ mods q.

- The polynomial $f^{-1}(x)$ can be computed using the *Extended Euclidean algorithm* for polynomials.
- The public key is $h(x)$ and the private key is $f(x)$.
- The polynomial $g(x)$ is used in the construction of the public key $h(x)$; $g(x)$ is not part of the public or private key, but it should be kept secret and then discarded after $h(x)$ is formed.

NTRU Encryption and Decryption

- A plaintext is a polynomial $m(x) \in \mathcal{R}_q$ in which every coefficient is in the set $\{-1, 0, 1\}$. To encrypt $m(x)$, perform the following operations.
  1. First, choose a polynomial $r(x) \in \mathcal{R}_q$ in which every coefficient is in the set $\{-1, 0, 1\}$. Usually, it is required that there is a certain distribution of the values $-0, 1, -1$ among the coefficients of $r(x)$.
  2. The ciphertext $y(x)$ is computed as

$$y(x) = r(x)h(x) + m(x) \text{ mods } q.$$

- To decrypt a ciphertext $y(x)$, perform the following operations:
  1. Compute $a(x) = f(x)y(x) \text{ mods } q$.
  2. Compute $m'(x) = a(x) \text{ mods } p$. Notice that this final computation is a centred reduction modulo p.
- If all goes well, it will be the case that $m'(x) = m(x)$.

NTRU Decryption (cont.)

- It is easy to verify that the following equations hold in $\mathcal{R}_q$ (i.e., they are congruences modulo q and modulo $x^N - 1$):

$$\begin{aligned} a(x) &= f(x)y(x) \\ &= f(x)(r(x)h(x) + m(x)) \\ &= f(x)(r(x)f^{-1}(x)g(x) + m(x)) \\ &= r(x)g(x) + f(x)m(x). \end{aligned} \tag{1}$$

- Suppose that this congruence (1) is actually an equality in $\mathcal{R}$. This happens if and only if every coefficient of $r(x)g(x) + f(x)m(x)$ lies in the interval

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

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

NTRU Decryption

- Now, assuming that the equation (1) holds in $\mathcal{R}$ and reducing modulo p, we have

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

From this relation, we see that

$$m(x) = a(x) \text{ mods } p,$$

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

A Lattice-based Attack

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

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

A Lattice-based Attack (cont.)

- Recall that
$$f(x)h(x) \equiv g(x) \pmod{q},$$
so
$$f(x)h(x) - g(x) = q\,t(x)$$
for some polynomial $t(x) \in \mathbb{Z}[x]$.
- Therefore the lattice $\mathcal{L}_{\mathbf{h}}$ consists of the following vectors:
$$\mathcal{L}_{\mathbf{h}} = \{(\mathbf{a}, \mathbf{b}) \in \mathbb{Z}^{2N} : a(x)h(x) \equiv b(x) \pmod{q}\}.$$
- It is then straightforward to compute
$$(\mathbf{f}, -\mathbf{t})M = (\mathbf{f}, \mathbf{g}),$$
so $(\mathbf{f}, \mathbf{g}) \in \mathcal{L}_{\mathbf{h}}$.
- The vector $(\mathbf{f}, \mathbf{g})$ has a small norm, so it is plausible that $(\mathbf{f}, \mathbf{g})$ (or a cyclic rotation of this vector) is the shortest vector in the lattice $\mathcal{L}_{\mathbf{h}}$.
- If this instance of the **Shortest Vector** problem can be solved, the adversary might find the private key $f(x)$.

Example

- Suppose $N = 13$, $q = 257$ and $p = 3$. Here is an NTRU public key:

$$\begin{aligned} h(x) =& 123x^{12} - 88x^{11} + 13x^{10} - 30x^9 + 30x^8 + 10x^7 - 103x^6 \\ & - 81x^5 - 93x^4 + 94x^3 - 98x^2 - 68x - 96. \end{aligned}$$

- This key was constructed from the two polynomials

$$g = 3x^{11} - 3x^{10} + 3x^9 + 3x^3 - 3x$$

and

$$f = 3x^{11} - 3x^9 + 3x^8 - 3x^4 - 3x^2 + 1.$$

- The matrix M is a 26 by 26 matrix, given on the next slide.

The Matrix M

$$M := \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -96 & -68 & -98 & 94 & -93 & -81 & -103 & 10 & 30 & -30 & 13 & -88 & 123 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 123 & -96 & -68 & -98 & 94 & -93 & -81 & -103 & 10 & 30 & -30 & 13 & -88 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -88 & 123 & -96 & -68 & -98 & 94 & -93 & -81 & -103 & 10 & 30 & -30 & 13 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & -88 & 123 & -96 & -68 & -98 & 94 & -93 & -81 & -103 & 10 & 30 & -30 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -30 & 13 & -88 & 123 & -96 & -68 & -98 & 94 & -93 & -81 & -103 & 10 & 30 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 30 & -30 & 13 & -88 & 123 & -96 & -68 & -98 & 94 & -93 & -81 & -103 & 10 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 10 & 30 & -30 & 13 & -88 & 123 & -96 & -68 & -98 & 94 & -93 & -81 & -103 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -103 & 10 & 30 & -30 & 13 & -88 & 123 & -96 & -68 & -98 & 94 & -93 & -81 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -81 & -103 & 10 & 30 & -30 & 13 & -88 & 123 & -96 & -68 & -98 & 94 & -93 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -93 & -81 & -103 & 10 & 30 & -30 & 13 & -88 & 123 & -96 & -68 & -98 & 94 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 94 & -93 & -81 & -103 & 10 & 30 & -30 & 13 & -88 & 123 & -96 & -68 & -98 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -98 & 94 & -93 & -81 & -103 & 10 & 30 & -30 & 13 & -88 & 123 & -96 & -68 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -68 & -98 & 94 & -93 & -81 & -103 & 10 & 30 & -30 & 13 & -88 & 123 & -96 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 257
\end{bmatrix}$$

The LLL-reduced basis

$$\begin{bmatrix}
0 & 0 & 3 & -3 & 0 & 3 & 0 & 1 & 0 & -3 & 0 & -3 & 0 & 0 & 0 & 0 & 3 & -3 & 3 & 0 & 0 & -3 & 0 & 3 & 0 & 0 \\
0 & 3 & 0 & 1 & 0 & -3 & 0 & -3 & 0 & 0 & 0 & 3 & -3 & -3 & 3 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 3 & -3 & 0 & 3 & 0 & 1 & 0 & -3 & 0 & -3 & 0 & 0 & 0 & 0 & 3 & -3 & 3 & 0 & 0 & -3 & 0 & 3 & 0 \\
-3 & 0 & 3 & 0 & 1 & 0 & -3 & 0 & -3 & 0 & 0 & 0 & 3 & 3 & -3 & 3 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 3 & 0 & 0 & 0 & -3 & 3 & 0 & -3 & 0 & -1 & 3 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & -3 & 0 & 0 \\
0 & -3 & 3 & 0 & -3 & 0 & -1 & 0 & 3 & 0 & 3 & 0 & 0 & 0 & 0 & -3 & 3 & -3 & 0 & 0 & 3 & 0 & -3 & 0 & 0 & 0 \\
0 & 1 & 0 & -3 & 0 & -3 & 0 & 0 & 0 & 3 & -3 & 0 & 3 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 3 & -3 & 3 \\
3 & 0 & 0 & 0 & -3 & 3 & 0 & -3 & 0 & -1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & -3 & 0 & 0 & 3 & 0 & -3 \\
-1 & 0 & 3 & 0 & 3 & 0 & 0 & 0 & -3 & 3 & 0 & -3 & 0 & 0 & 3 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & -3 & 0 \\
0 & -3 & 0 & 0 & 0 & 3 & -3 & 0 & 3 & 0 & 1 & 0 & -3 & 3 & 0 & 0 & 0 & 0 & 0 & 3 & -3 & 3 & 0 & 0 & -3 & 0 \\
3 & 0 & 3 & 0 & 0 & 0 & -3 & 3 & 0 & -3 & 0 & -1 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & -3 & 0 & 0 & 3 \\
-3 & 3 & 0 & -3 & 0 & -1 & 0 & 3 & 0 & 3 & 0 & 0 & 0 & 0 & -3 & 3 & -3 & 0 & 0 & 3 & 0 & -3 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & -3 & 0 & -3 & 0 & 0 & 0 & 3 & -3 & 0 & 3 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 3 & -3 \\
-14 & -8 & 7 & 0 & -7 & -12 & -1 & -12 & -5 & -4 & -2 & -5 & -6 & -6 & -12 & 0 & -1 & 3 & 6 & 0 & -6 & 9 & 0 & -6 & -3 & -9 \\
5 & 10 & -2 & 9 & 3 & 5 & 6 & 3 & 1 & 11 & 2 & -6 & 4 & -3 & -6 & 3 & 12 & -3 & -3 & 6 & 6 & 9 & 9 & 18 & 3 & 1 \\
5 & 0 & 2 & 3 & 8 & 5 & -4 & 3 & 8 & 12 & -2 & 15 & 6 & -3 & 3 & 3 & 12 & 12 & 9 & 3 & 1 & 0 & -9 & 3 & 9 & -6 \\
10 & -4 & 12 & 3 & -1 & 3 & 5 & 1 & 15 & 5 & -9 & 4 & 5 & -6 & 3 & 6 & 0 & 0 & 3 & 9 & 12 & 9 & 12 & 6 & 1 & 0 \\
-6 & -2 & -6 & -5 & -1 & -12 & -5 & 6 & -1 & -2 & -9 & 5 & -15 & 3 & 3 & -3 & -9 & -12 & -9 & -9 & -6 & -1 & 3 & 6 & -6 & -9 \\
1 & 12 & 5 & -12 & 4 & 5 & 10 & -1 & 9 & 3 & 2 & 3 & 6 & 9 & 9 & 15 & 6 & 1 & 0 & -6 & 3 & 9 & -3 & 3 & 3 & 9 \\
9 & -1 & -2 & -7 & 2 & -9 & -6 & -5 & -3 & -6 & -4 & -12 & -3 & 0 & -4 & 0 & 6 & -3 & -12 & 3 & 0 & -6 & -6 & -12 & -9 & -15 \\
8 & -7 & 0 & 7 & 12 & 1 & 12 & 5 & 4 & 2 & 5 & 6 & 14 & 12 & 0 & 1 & -3 & -6 & 0 & 6 & -9 & 0 & 6 & 3 & 9 & 6 \\
-2 & -3 & -8 & -5 & 4 & -3 & -8 & -12 & 2 & -15 & -6 & -5 & 0 & -3 & -12 & -12 & -9 & -3 & -1 & 0 & 9 & -3 & -9 & 6 & 3 & -3 \\
3 & 7 & 9 & 1 & 11 & 5 & 7 & 2 & 8 & 6 & 14 & 8 & -10 & -2 & 0 & -9 & 0 & 6 & -6 & 0 & 3 & 3 & 9 & 6 & 12 & 0 \\
-9 & -6 & -5 & -3 & -6 & -4 & -12 & -3 & 9 & -1 & -2 & -7 & 2 & -12 & 3 & 0 & -6 & -6 & -12 & -9 & -15 & 0 & -4 & 0 & 6 & -3 \\
-2 & 12 & 2 & 4 & 2 & 5 & 9 & 11 & 8 & -4 & 0 & 8 & 12 & 0 & 9 & -9 & 0 & 6 & 3 & 9 & 9 & 9 & 3 & 1 & -3 & -9 \\
6 & 6 & 1 & 12 & 2 & -9 & 4 & 2 & 10 & -2 & 9 & 6 & 2 & 3 & 9 & 9 & 9 & 15 & 3 & 4 & -3 & -6 & 3 & 12 & -3 & 0
\end{bmatrix}$$

The LLL-reduced basis (cont.)

- The first vector in the LLL-reduced basis is

$$(0, 0, 3, -3, 0, 3, 0, 1, 0, -3, 0, -3, 0, 0, 0, 0, 3, -3, 3, 0, 0, -3, 0, 3, 0, 0).$$

- Split this vector into two halves:

$$(0, 0, 3, -3, 0, 3, 0, 1, 0, -3, 0, -3, 0), \ (0, 0, 0, 3, -3, 3, 0, 0, -3, 0, 3, 0, 0).$$

- Rotate these seven positions to the left:

$$(1, 0, -3, 0, -3, 0, 0, 0, 3, -3, 0, 3, 0), \ (0, -3, 0, 3, 0, 0, 0, 0, 0, 3, -3, 3, 0).$$

- These correspond to the polynomials

$$3x^{11} - 3x^9 + 3x^8 - 3x^4 - 3x^2 + 1 \quad \text{and} \quad 3x^{11} - 3x^{10} + 3x^9 + 3x^3 - 3x,$$

which are (respectively) $f(x)$ and $g(x)$.

Another Possible Attack

- An adversary could also attempt to decrypt a specific ciphertext, say $\mathbf{y}$.
- It turns out that this can be modelled as an instance of the **Closest Vector** problem.
- Thus the obvious attacks on *NTRU* involve natural lattice problems, but we do not have provable security (i.e., it is not proven that any successful attack on *NTRU* breaks a certain lattice problem via a reduction).
- However, there are examples of lattice-based cryptography (both encryption and signature schemes) that can be proven secure if certain lattice problems are intractable.

Learning with Errors

Given a prime q, it is possible to solve systems of linear equations in n variables over $\mathbb{Z}_q$ efficiently. If we introduce noise into the system, we obtain the **Learning With Errors** (or **LWE**) problem, which is believed to be difficult to solve. **LWE** was introduced by Regev in 2005.

Learning With Errors

Instance:

1. Positive integers m, n, β and a prime q with $\beta \ll q$.
2. An m by n matrix A with entries in $\mathbb{Z}_q$.
3. A vector $\mathbf{b} = (b_1, \ldots, b_m) \in (\mathbb{Z}_q)^m$, which is computed as

$$\mathbf{b}^T = A\mathbf{s}^T + \mathbf{e}^T \bmod q,$$

where $\mathbf{s} \in (\mathbb{Z}_q)^n$ and $\mathbf{e}$ are secret vectors such that $\|\mathbf{e}\|_\infty \leq \beta$ when the coordinates of $\mathbf{e}$ are reduced mods q.

Find: The secret vector $\mathbf{s}$ (given A and $\mathbf{b}$).

Learning with Errors (cont.)

- **LWE** can be regarded as the problem of finding a solution modulo q to an approximate system of linear equations.
- It can be viewed as a modified **Closest Vector** problem.
- Also, it closely resembles the problem of decoding a received vector in a linear code, except we are now in the setting of a lattice.
- Constructing an **LWE** system that is difficult to solve requires a value of $q \gg n$ as well as a careful choice of a probability distribution χ that is used to choose the error vector $\mathbf{e}$.
- Various proposals use a Gaussian or a uniform distribution.
- The **LWE** problem is of interest to cryptographers because
  1. there is a reduction from the **Gap Shortest Vector** problem to **LWE**,
  2. **LWE** is self-reducible (so its average-case and worst-case complexity are similar).
- **Gap Shortest Vector** is believed to be difficult to solve in the worst-case, so **LWE** is (probably) difficult to solve in the average-case.

Regev Cryptosystem

Construct an appropriate instance of the **Learning With Errors** problem.
The private key is the vector $\mathbf{s} \in (\mathbb{Z}_q)^n$.
The public key consists of the matrix A and the vector $\mathbf{b}$.

To encrypt a one-bit message x, choose a $\{0,1\}$-vector $\mathbf{r}$ of length n.
The ciphertext $y = (\mathbf{y_1}, y_2)$ is given by

$$y = \begin{cases} (\mathbf{r}A, \mathbf{r}\mathbf{b}^T) & \text{if } x = 0, \\ (\mathbf{r}A, \mathbf{r}\mathbf{b}^T + \lfloor \frac{q}{2} \rfloor) & \text{if } x = 1. \end{cases}$$

To decrypt a ciphertext $(\mathbf{y_1}, y_2)$, compute the quantity

$$z = y_2 - \mathbf{y_1}\mathbf{s}^T \bmod q.$$

The decrypted message is 0 if z closer to 0 than $\lfloor q/2 \rfloor$, and 1 otherwise. That is, we decrypt y to 0 if $z \in [0, \lfloor q/4 \rfloor) \cup [\lfloor 3q/4 \rfloor, q-1]$ and we decrypt y to 1 if $z \in [\lceil q/4 \rceil, \lfloor 3q/4 \rfloor)$.

Example

Let $m = n = 3$ and $q = 11$. Suppose that each coordinate of $\mathbf{e}$ takes on each of the values 0, 1, or -1 with probability $1/3$. Thus $\mathbf{e} = (0, 1, -1)$ is a typical choice. Suppose the private key is $\mathbf{s} = (1, 2, 3)$ and

$$A = \begin{pmatrix} 5 & 8 & 10 \\ 4 & 9 & 1 \\ 3 & 6 & 0 \end{pmatrix},$$

so

$$\mathbf{b}^T = A\mathbf{s}^T + \mathbf{e}^T = \begin{pmatrix} 7 \\ 4 \\ 3 \end{pmatrix}.$$

The public key consists of A and $\mathbf{b}$.

Example

To encrypt the plaintext $x = 1$, we choose a random $\{0,1\}$-vector, say $\mathbf{r} = (1,0,1)$. Then the ciphertext is $(\mathbf{y_1}, y_2)$ where

$$\mathbf{y_1} = (1,0,1)\begin{pmatrix} 5 & 8 & 10 \\ 4 & 9 & 1 \\ 3 & 6 & 0 \end{pmatrix} \bmod 11 = (8,3,10)$$

$$y_2 = (1,0,1)\begin{pmatrix} 7 \\ 4 \\ 3 \end{pmatrix} + \lfloor 11/2 \rfloor \bmod 11 = 10 + 5 \bmod 11 = 4.$$

To decrypt the ciphertext $((8,3,10),4)$, we compute the following in $\mathbb{Z}_{11}$:

$$4 - (8,3,10)\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = 4 - (8+6+30) \bmod 11 = 4.$$

Because 4 is closer to $\lfloor \frac{11}{2} \rfloor = 5$ than to 0, the decrypted message is 1.

Ring LWE

- the *Regev Cryptosystem* has been proven to be secure against chosen plaintext attacks (i.e., it is CPA-secure) if **LWE** is intractable.
- The *Regev Cryptosystem* is not practical due to the large overhead required to encrypt a single bit, but there are modifications that are more efficient.
- Recall that **LWE** can be expressed using matrix notation as follows:
  1. Let $\mathbf{b}^T = A\mathbf{s}^T + \mathbf{e}^T$, where $A = (a_{i,j})$ is an m by n matrix over $\mathbb{Z}_q$ and $\mathbf{b}$, $\mathbf{s}$ and $\mathbf{e}$ are vectors of length n over $\mathbb{Z}_q$.
  2. Then, given A and $\mathbf{b}$ (but not $\mathbf{e}$), the goal is to find $\mathbf{s}$.
- In the **Ring LWE** problem, we work in the ring $\mathbb{Z}_q[x]/(x^n + 1)$ instead of $(\mathbb{Z}_q)^n$.
- In this ring, $x^n = -1$, $x^{n+1} = -x$, etc.
- Suppose $m = n$. We can think of A and $\mathbf{s}$ (in the associated **LWE** problem) as corresponding to polynomials $a(x)$ and $s(x)$.
- Computing $a(x)s(x) \bmod (x^n + 1)$ is equivalent to computing a matrix product $A\mathbf{s}^T$ where A has a special structure.

Example

- Suppose $m = n = 3.$ and $q = 11.$ Let $a(x) = 3x^2 + 7x + 5$ and $s(x) = 4x^2 + 7.$
- The associated A and $\mathbf{s}$ are as follows:

$$A = \begin{pmatrix} 5 & 7 & 3 \\ 8 & 5 & 7 \\ 4 & 8 & 5 \end{pmatrix} \quad \text{and} \quad \mathbf{s} = (4, 0, 7).$$

- A is a (modified) circulant matrix with the following wrapping rule: replace any entry a_i by $-a_i$ when it wraps around.
- The first row of A is the polynomial $a(x)$ with low degree terms preceding high degree terms.
- $\mathbf{s}$ is written with the high degree terms first.

Example

- The matrix computation (in $\mathbb{Z}_{11}$) is

$$A\mathbf{s}^T = \begin{pmatrix} 5 & 7 & 3 \\ 8 & 5 & 7 \\ 4 & 8 & 5 \end{pmatrix} \begin{pmatrix} 4 \\ 0 \\ 7 \end{pmatrix} = \begin{pmatrix} 8 \\ 4 \\ 7 \end{pmatrix}.$$

- The polynomial computation, using the reduction $x^3 = -1 = 10$, is

$$\begin{aligned} a(x)s(x) &= (3x^2 + 7x + 5)(4x^2 + 7) \\ &= 12x^4 + 21x^2 + 28x^3 + 49x + 20x^2 + 35 \\ &= x^4 + 10x^2 + 6x^3 + 5x + 9x^2 + 2 \\ &= 10x + 10x^2 + 5 + 5x + 9x^2 + 2 \\ &= 8x^2 + 4x + 7. \end{aligned}$$

- So the same result is obtained by both methods.

Discussion

- In the **Ring LWE** setting, it is only necessary to specify the first row of A, since all the other rows can be derived from the first row.
- Since A is part of the public key, using **Ring LWE** in place of **LWE** reduces the size of the public key very significantly.
- There are other possible computational speedups that can be obtained by using efficient algorithms for polynomial multiplication.
- Another advantage of the **Ring LWE** setting is that it supports encryption of binary n-tuples.
- However, the lattice associated with the matrix A is now more structured than it was when A was random, which could conceivably affect security.
- We describe the *LPR* cryptosystem, due to Lyubashevsky, Peikert and Regev, from EUROCRYPT 2010.
- *LPR* is secure if the associated **Ring LWE** problem is intractable.

LPR Ring-LWE Cryptosystem (key generation)

Let $R = \mathbb{Z}[x]/(x^n + 1)$, let q be a large prime, and let $R_q = R \bmod q$. Also let β be a small positive integer and define $B = \{0, \ldots, \beta - 1\}$.

The private key is a polynomial $s(x) \in R$ having coefficients in B.

The public key consists consists of two polynomials $a(x), b(x) \in R_q$ where

- $a(x)$ is chosen uniformly at random from R_q,
- $e(x)$ is a (secret) polynomial in R having coefficients in B, and
- $b(x) = a(x)s(x) + e(x) \bmod q$.

A plaintext is a polynomial $z(x) \in R$ in which all coefficients are equal to 0 or 1, so we can think of z as being equivalent to an n-bit plaintext.

LPR Ring-LWE Cryptosystem (encryption and decryption)

To encrypt $z(x)$, choose three polynomials $r(x), e_1(x), e_2(x) \in R$ having coefficients in B. The ciphertext y consists of two polynomials $u(x), v(x) \in R_q$, computed as follows:

$$u(x) = a(x)r(x) + e_1(x) \bmod q$$

$$v(x) = b(x)r(x) + e_2(x) + \left\lfloor \frac{q}{2} \right\rfloor z(x) \bmod q.$$

To decrypt a ciphertext $(u(x), v(x))$, compute the polynomial

$$v(x) - u(x)s(x) \text{ mods } q.$$

Each coefficient of the plaintext is deemed to be 0 if the absolute value of the corresponding coefficient of $v(x) - u(x)s(x)$ is $\leq \lfloor q/4 \rfloor$, and 1 otherwise.

Kyber-PKE

- *Kyber-PKE* (which was standardized by NIST in 2024 as FIPS 203) can be thought of as an extension of the *LPR* cryptosystem.
- One main difference between *LPR* and *Kyber-PKE* is that *Kyber-PKE* works in the setting of modules rather than rings.
- More precisely, we perform computations on tuples of polynomials.
- Various optimizations are also incorporated into the *Kyber-PKE*.
- *Kyber-PKE* is a public-key cryptosystem secure against a chosen-plaintext attack, but this is not sufficient for its intended use as a key encapsulation mechanism.
- A strengthened version of *Kyber-PKE* is used to obtain the key encapsulation mechanism known as *ML-KEM* (or *Module-Lattice-Based KEM*), which is secure against a chosen-ciphertext attack.
- The technique used to construct the KEM is based on the *Fujisaki-Okamoto Transform*, which is a standard method of strengthening a CPA-secure cryptosystem to be CCA-secure.

Dilithium Signature Scheme

- *Dilithium* is a lattice-based signature scheme.
- It has been standardized by NIST as the *Module-Lattice-Based Digital Signature Standard* (FIPS 204).
- *Dilithium* has the same general structure as the *Schnorr Signature Scheme*.
- In particular, signing includes computing a commitment, a challenge (obtained by hashing the commitment and message) and a response.
- However, the module-based setting of *Dilithium* is similar to that of *Kyber-PKE*.
- The security of the private key depends on the intractability of a module-based **Decision LWE** problem.
- It is also infeasible to forge a signature if a module-based non-homogeneous **Short Integer Solutions** problem is intractable (**SIS** can be viewed intuitively as a modified **Shortest Vector** problem).

Coding Theory vs Lattice Theory

There are many "analogous" concepts in coding theory and lattice theory.

coding theory	lattice theory
vector space over a finite field	$\mathbb{Z}$-module
hamming weight/distance	Euclidean or infinity norm
minimum-weight codeword	shortest vector (also **SIS**)
closest codeword (decoding problem)	closest vector (also **LWE**)

Even the design of some public-key cryptosystems is similar in the two settings:

- encryption: encode a vector, add noise
- decryption: decode the ciphertext (remove the noise)
- perform computations in a polynomial ring
- public key is the quotient of two secret polynomials (*NTRU*, *BIKE*)
- etc.

5 Other approaches to post-quantum cryptography

- Additional digital signature proposals
- Multivariate quadratic equations
- Oil-and-Vinegar
- Isogeny-based cryptography
- Resources

NIST Standardization

- As mentioned in Part 1, NIST selected 14 candidate algorithms in October 2024 to move forward to the second round of evaluation of Additional Digital Signature Proposals.

- The techniques used in the design of the 14 candidate signature schemes can be categorized as follows:

  1. **code-based:** *CROSS* and *LESS*
  2. **isogeny-based:** *SQIsign*
  3. **lattice-based:** *HAWK*
  4. **MPC-in-the-head:** *Mirath*, *MQOM*, *PERK*, *RYDE* and *SDitH*
  5. **multivariate:** , *MAYO*, *QR-UOV*, *SNOVA*, and *UOV*
  6. **symmetric cryptography-based:** *FAEST*.

- Here I will give a very brief introduction to multivariate cryptography (specifically, the *Oil-and-Vinegar* signature scheme) and isogeny-based cryptography (*SIDH*, which is actually a broken key agreement scheme).

Multivariate Quadratic Equations

- Another problem suggested for the design of post-quantum cryptosystems is that of finding solutions to large systems of quadratic equations in many variables over a finite field.
- This is the **Multivariate Quadratic Equations** (or **MQ**) problem.

Multivariate Quadratic Equations

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

$$f_k(x_1, x_2, \ldots, x_n) = d_k,$$

$1 \leq k \leq m$, where each polynomial f_k has the form

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

with $a_{ij}, b_i, c \in \mathbb{F}_q$, for all i, j.

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

Strategy

- The **MQ** problem is NP-hard over any finite field.
- The design of cryptosystems based on the **MQ** problem follows a similar strategy to that of several other public-key cryptosystems.
- Namely, it involves starting with a special case of the problem that is easy to solve, and then disguising it with the aim of making it appear like a general instance of the problem.
- *Hidden Field Equations* was a public-key cryptosystem, based on the *MQ* problem, that was broken.
- Signature schemes based on *MQ* currently seem to be more promising.
- The original *Oil-and-Vinegar* signature scheme was proposed by Patarin in 1997.
- An unbalanced version of *OV* is under consideration in the second round of the NIST evaluation of additional digital signature proposals.

Oil-and-Vinegar

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

Oil-and-Vinegar Set-up

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

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

  for $1 \leq k \leq n$.
- No terms in any f_k involve the product of two oil variables.
- Given $(m_1, m_2, \ldots, m_n) \in (\mathbb{F}_q)^n$, we can find a solution to the following system of multivariate quadratic equations:

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

- To do this, we choose random values $v_1, v_2, \ldots, v_n \in \mathbb{F}_q$ for the vinegar variables.

Oil-and-Vinegar Set-up (cont.)

- We thus obtain a system of n linear equations in the n oil variables, which we can then (hopefully) solve to find a solution

$$(v_1, v_2, \ldots, v_n, o_1, o_2, \ldots, o_n).$$

- In the case where the system of linear equations has no solutions, we try new values for the vinegar variables, until we find a system that does have solutions.
- In order to disguise the special nature of the signing polynomials, we "mix" the oil and vinegar variables with the use of an affine transformation $S : (\mathbb{F}_q)^{2n} \to (\mathbb{F}_q)^{2n}$ defined by

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

where M is a $2n \times 2n$ invertible matrix and $(r_1, r_2, \ldots, r_{2n}) \in (\mathbb{F}_q)^{2n}$ is a random vector.

Oil-and-Vinegar Set-up (cont.)

- Note that the inverse transformation is simply

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

- The affine transformation S allows us to define public verification polynomials f_k^{pub} that appear to be more complicated than the private signing polynomials that are used to compute the signature.
- The private signing key consists of S and the n polynomials f_k, $1 \leq k \leq n$.
- The public verification key consists of polynomials defined as follows:

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

$k = 1, \ldots, n$.

Oil-and-Vinegar: Signing and Verification

- A message (or more likely, a message digest) $(m_1, \ldots, m_n)$ is signed as follows:
  1. First, find a solution $(v_1, v_2, \ldots, v_n, o_1, o_2, \ldots, o_n)$ to the system of equations $f_k(x_1, x_2, \ldots, x_{2n}) = m_k$ for all k with $1 \leq k \leq n$.
  2. The inverse transformation S^{-1} is then applied to obtain the signature
  $$(s_1, s_2, \ldots, s_{2n}) = S^{-1}(v_1, v_2, \ldots, v_n, o_1, o_2, \ldots, o_n).$$
- Verifying a signature is done as follows:
- A valid signature on a message $(m_1, m_2, \ldots, m_n) \in (\mathbb{F}_q)^n$ is a vector $(s_1, s_2, \ldots, s_{2n}) \in (\mathbb{F}_q)^{2n}$ such that
$$f_k^{\text{pub}}(s_1, s_2, \ldots, s_{2n}) = m_k,$$
$k = 1, \ldots, n$.

Example

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

$$f_1(x_1, x_2, x_3, x_4) = x_1x_2 + x_2x_3 + x_4,$$
$$f_2(x_1, x_2, x_3, x_4) = x_1x_3 + x_2x_4 + x_2.$$

Let S be the affine transformation that maps

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

S^{-1} is the transformation that maps

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

The polynomials f_1^{pub} and f_2^{pub} are given by

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

Example

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

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

- This system has the unique solution $x_3 = x_4 = 0$, and hence $(0, 1, 0, 0)$ is a solution to our original system of equations.
- The signature is $S^{-1}(0, 1, 0, 0) = (0, 1, 1, 0)$.
- To verify that $(0, 1, 1, 0)$ is a valid signature for the message $(0, 1)$, we compute $f_1^{\text{pub}}(0, 1, 1, 0) = 0$ and $f_2^{\text{pub}}(0, 1, 1, 0) = 1$.

Comments

- The original *Oil and Vinegar Signature Scheme* was broken; the Kipnis-Shamir attack (using linear algebra) can be used to find the private key or an "equivalent" key.
- Various extensions (e.g., *Rainbow*) were also broken.
- One promising approach to improving the security of this scheme is to increase the number of vinegar variables, yielding the so-called *Unbalanced Oil and Vinegar Signature Scheme*.
- This scheme was proposed by Kipnis, Patarin and Goubin in 1999.
- One suggested parameter set (under consideration for the second round of the NIST evaluation of additional digital signature proposals) is defined over $\mathbb{F}_{256}$ and has 44 oil variables and 68 vinegar variables.
- We should mention that Gröbner basis algorithms can be used to attempt to forge signatures, so any MQ-based scheme must take these kinds of attacks into account.

Elliptic Curve Isogenies

- A rational map is a function ϕ that maps points of one elliptic curve $\mathcal{E}_1$ to points of another elliptic curve $\mathcal{E}_2$.
- It is required that ϕ has the form

$$\phi(x, y) = \left(\frac{p_1(x, y)}{q_1(x, y)}, \frac{p_2(x, y)}{q_2(x, y)} \right)$$

  where p_1, p_2, q_1, q_2 are polynomials in x and y.
- An isogeny is a rational map that is a group automorphism (i.e., $\phi(P) + \phi(Q) = \phi(P + Q)$, where "+" denotes addition on the relevant elliptic curve.
- One example of an isogeny from an elliptic curve to itself is multiplication by an integer constant c, i.e., the mapping $P \mapsto cP$.
- This mapping is obviously a homomorphism, but it is also a rational map, which is perhaps less obvious.

SIDH

- *Supersingular Isogeny-Based Diffie-Hellman* (or *SIDH*) is a modification of *Diffie-Hellman* key agreement that is built from isogenies.
- It uses supersingular elliptic curves.
- *SIDH* was introduced by Jao and De Feo in 2011.
- *Supersingular Isogeny Key Encapsulation* (or *SIKE*) is a key encapsulation mechanism based on *SIDH*.
- It was chosen as a round 4 candidate for the NIST standardization of post-quantum cryptography, but a fatal attack on it was described in August 2022.
- See the paper An efficient key recovery attack on SIDH by Castryck and Decru (EUROCRYPT 2023).

Elliptic Curve Diffie-Hellman and SIDH

- Recall how a basic *Diffie-Hellman key exchange* would be carried out on an elliptic curve $\mathcal{E}$.
  1. P is a fixed point on $\mathcal{E}$.
  2. Alice chooses n_A and computes $n_A P$, which she sends to Bob.
  3. Bob chooses n_B and computes $n_B P$, which he sends to Alice.
  4. Alice computes $n_A(n_B P)$ and Bob computes $n_B(n_A P)$. Thus Alice and Bob both have the shared secret $n_A n_B P$.
- Very roughly speaking, *SIDH* replaces the point multiples $n_A P$ and $n_B P$ by appropriate elliptic curve isogenies.
  1. $\mathcal{E}$ is a fixed supersingular elliptic curve.
  2. Alice chooses an isogeny ϕ_A and computes $\phi_A(\mathcal{E}) = \mathcal{E}_A$, which she sends to Bob.
  3. Bob chooses an isogeny ϕ_B and computes $\phi_B(\mathcal{E}) = \mathcal{E}_B$, which he sends to Alice.
  4. Alice computes $\psi_A(\mathcal{E}_B)$ (where ψ_A is derived from ϕ_A) and Bob computes $\psi_B(\mathcal{E}_B)$ (where ψ_B is derived from ϕ_B). Alice and Bob both have computed the same elliptic curve.

Some Resources

A very incomplete list of references that discuss the mathematics underlying post-quantum cryptography:

- Shameless self-promotion: my forthcoming book A Primer on Post-quantum Cryptography should be published in early 2026.
- For detailed explanations of the new FIPS standards (especially the lattice-based schemes), see the Cryptography 101 lectures and lecture slides by Alfred Menezes on Youtube.
- Multivariate Public Key Cryptosystems, Second Edition by J. Ding, A. Petzoldt and D.S. Schmidt Springer, 2020, is a thorough treatment of the topic.
- Post-Quantum Cryptography, D.J. Bernstein, J. Buchmann and E. Dahmen (Eds.), is a classic reference published in 2009.

Thank you for your attention!