Post-Quantum Cryptography

Presented At:
5-Day Symposium and Training Workshop on
Quantum Information & Technologies (QIT)
Jointly Organized by IIIT Allahabad and C-DAC Pune
July 24-28, 2025

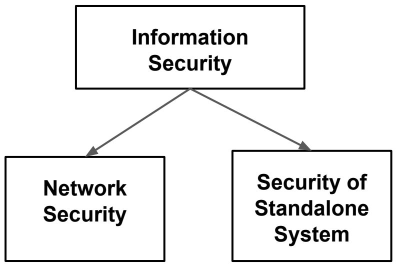
Presented By

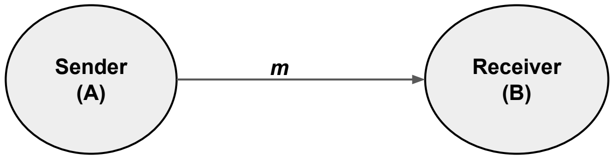
Dr. Soumyadev Maity, Assistant Professor
Department of Information Technology,
Indian Institute of Information Technology, Allahabad.

Overview

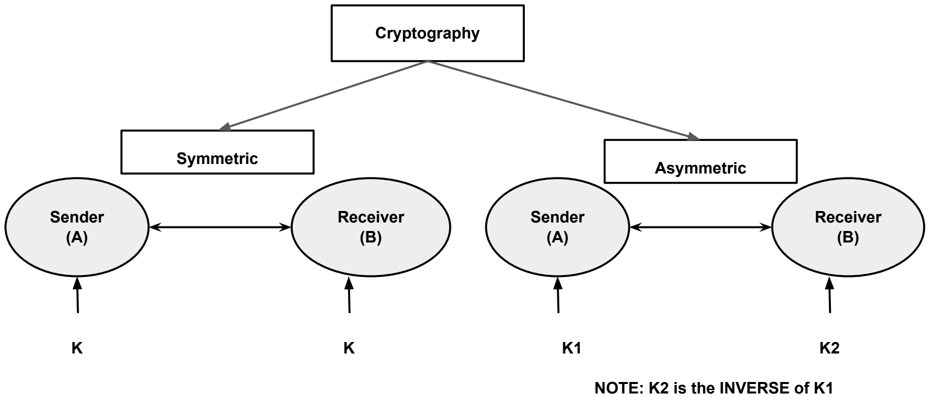
- **Objective of Information Security**
- **Fundamentals of Classical Cryptography**
- **Basics of Quantum Computing & Threat to Classical Cryptography**
- **Overview of Post-Quantum Cryptography**
- **Future Research Demands**

Objective of Information Security

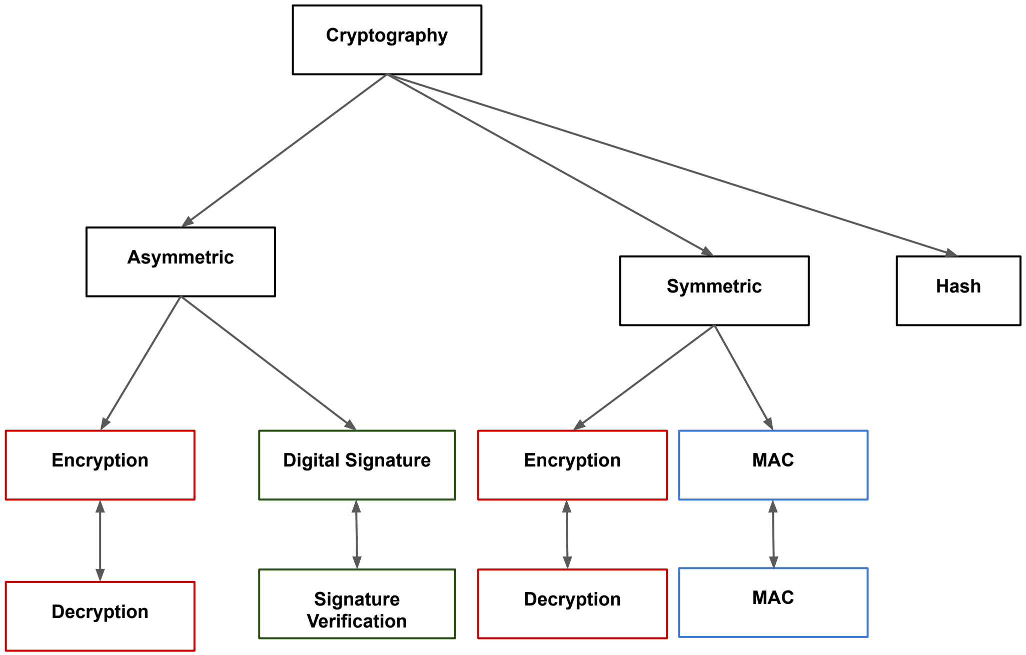




- What do we mean by secure Communication?
    - Basic Security Objectives:-
        - **Confidentiality (C)**
        - **Integrity (I)**
        - **Authentication (A)**
        - Non-Repudiation
    - Other Security Objectives:-
        - Availability
        - Authorization



Basic Cryptographic Techniques



Plaintext Key Ciphertext

$$E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$$

Ciphertext Key Plaintext

$$D : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$$

Message Digest

$$Hash : \{0,1\}^* \rightarrow \{0,1\}^l$$

Message Pri-Key Signature

$$Sig : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^l$$

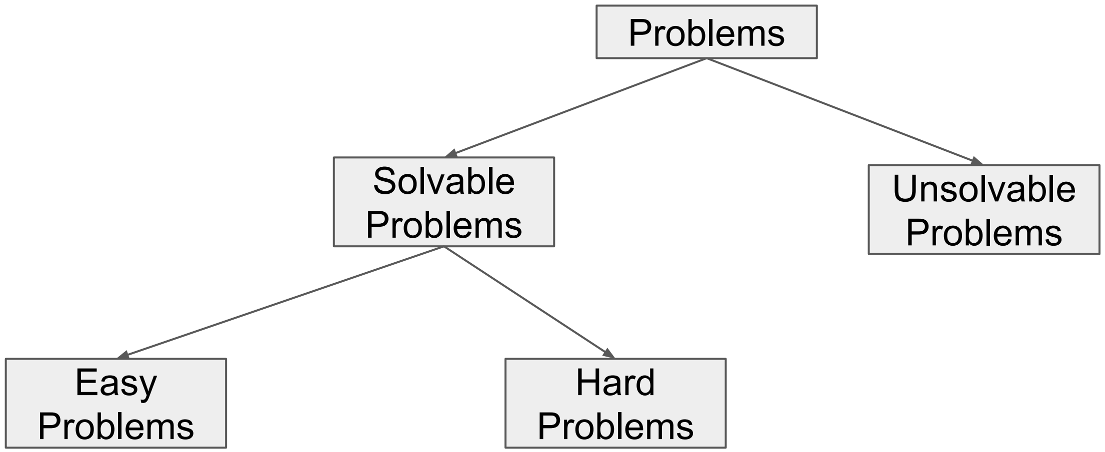
Message Pub-Key Signature

$$SigVerify : \{0,1\}^n \times \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}$$

Basic Cryptographic Techniques

	Confidentiality	Integrity	Authentication	Non-Repudiation
Symmetric Encryption-Decryption	YES	NO	YES	NO
Asymmetric Encryption-Decryption	YES	NO	YES	NO
Digital Signature & Verification	NO	YES	YES	YES
MAC Generation & Verification	NO	YES	YES	NO

Basic Cryptographic Techniques



Existing Most Efficient Algo. is Polynomial-time: O(k), O(k^2) etc.

- Existing Most Efficient Algo. is Not Polynomial-time: O(2^k), O(2^log(k)) etc.
- E.g.: N-Queen Problem, TSP problem, etc.

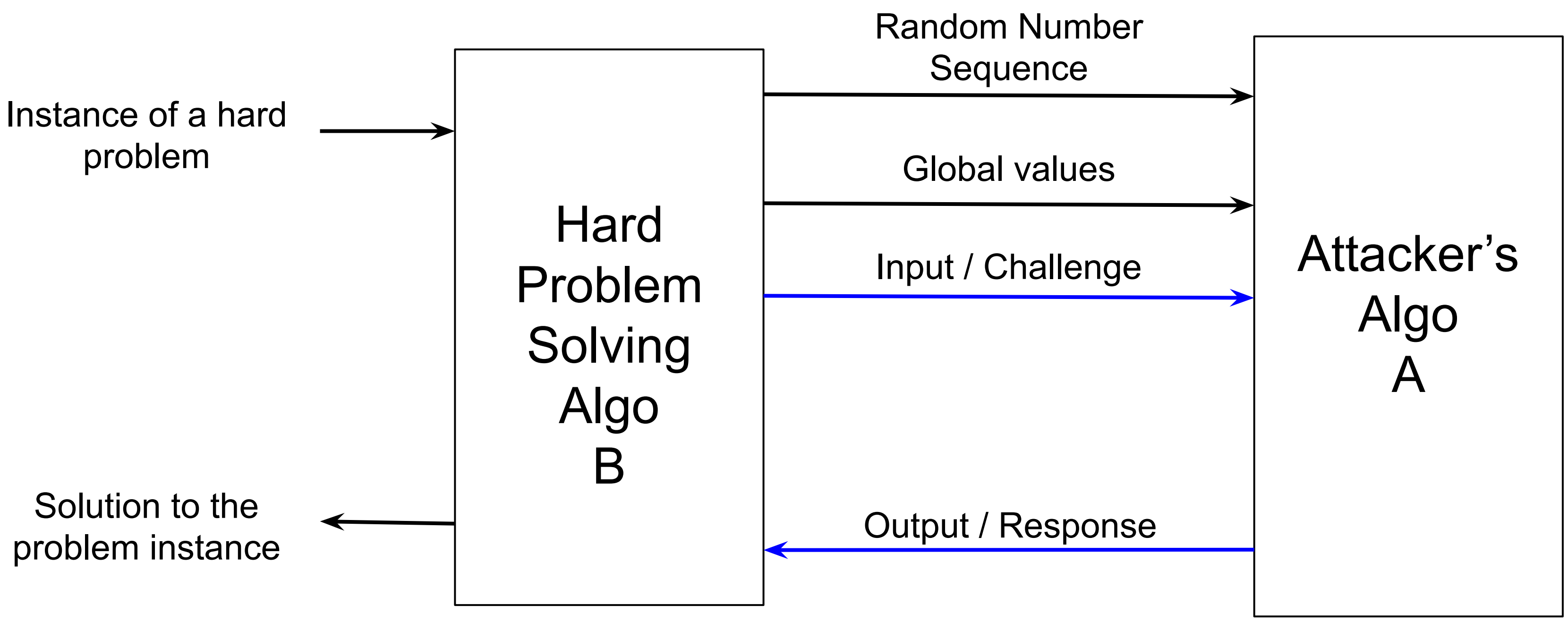
k : size of the input in no. of bits

- We need to show that Algo. **B** is also polynomial time
- We need to show that Algo. **B** has non-negligible probability of success
- Algo. **B** may call Algo. **A** polynomial no. of times
- Hard problems used in cryptography:
  - Integer factorization problem
  - Discrete log problem
  - Elliptic curve discrete log problem
  - Computational Diffie-Hellman Problem
  - Etc.
- Example: Integer factorization Problem:-

Suppose : n = p x q, where p, q are large primes

Problem 1:
Given: p, q
Compute: n
An Easy Problem

Problem 2:
Given: n
Compute: p (or q)
A Hard Problem



Proving Soundness of a Crypto. Scheme

Basics of Quantum Computing & Threat to Hard Problems

- **Basics of Quantum Computation**
    - **qubits**
    - **Superposition Theorem**
    - **Entanglement**
    - **No Cloning Theorem**
- **Shor's Algorithm**
    - **Uses Quantum Fourier Transformation to Find Order of an Element**
    - **Can solve integer factorization Problem in Polynomial Time**
    - **Can Solve Discrete Log Problem in Polynomial Time**

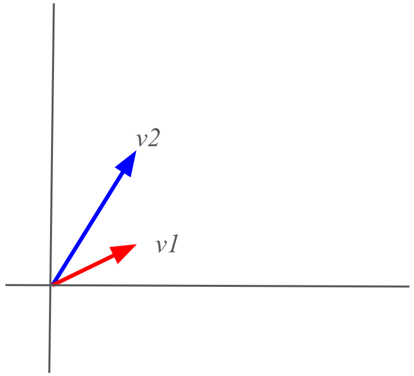
Basics of Quantum Computing & Threat to Hard Problems

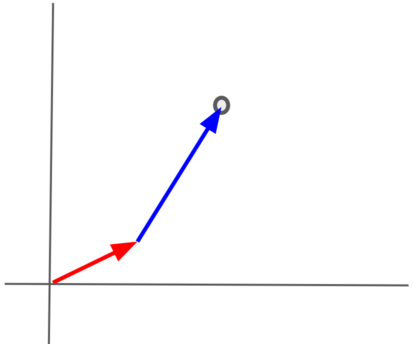
- **Public Key Cryptography - Depends on Hard Problem**
  - Not secure against Quantum Attacker
- **Symmetric Cryptography - Find Preimage Only Brute-Force Attack**
  - Grover's Algo: reduces search space quadratically $O(2^n)$ to $O(2^{(n/2)})$
- **Hash Functions - Find Collision using Brute-Force Attack**
  - Grover's Algo: reduces search space quadratically $O(2^{(n/2)})$ to $O(2^{(n/3)})$

Overview of Post-Quantum Cryptography

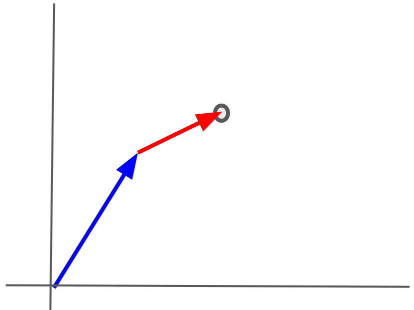
- **Post-Quantum Cryptography**
  - **Uses mathematical problem - assumed to be hard in presence of quantum computer**
    - **lattice based cryptography (**Used mathematical problems: shortest vector problem and learning with errors).
      - Example: NTRU, Kyber
    - **Code-based cryptography (**Used mathematical problems: decoding random linear codes)
      - Example: McEliece cryptosystem
    - **Multivariate polynomial cryptography:** utilizes the challenge of solving systems of multivariate polynomial equations.
      - Example: Rainbow, HFE (Hidden Field Equations)
    - **Hash-based cryptography:** uses difficulty of finding collisions in hash functions.
      - Example: XMSS (eXtended Merkle Signature Scheme), LMS (Leighton-Micali Signature)
    - **Isogeny-based cryptography:** Computing Isogenies Between Elliptic Curves: Finding a map between elliptic curves that preserves their structure.
      - Example: Supersingular Isogeny Key Encapsulation (SIKE)

Lattice based cryptography

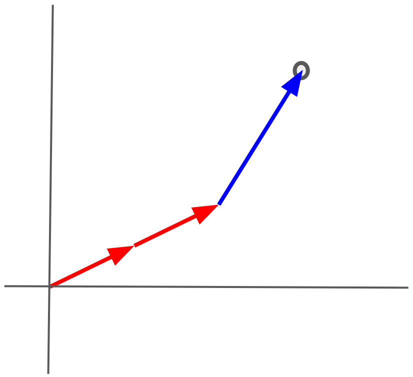




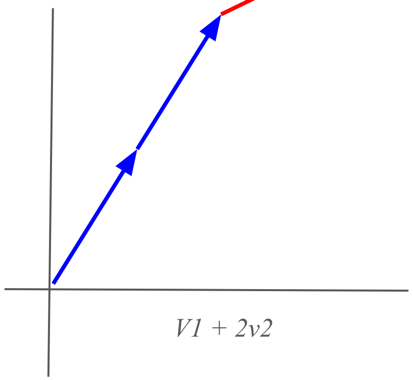
V1 + v2



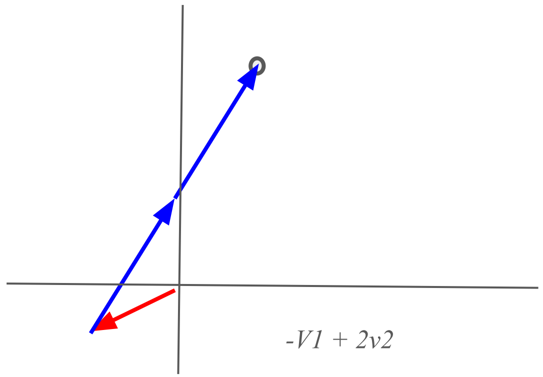
V2 + v1



2V1 + v2

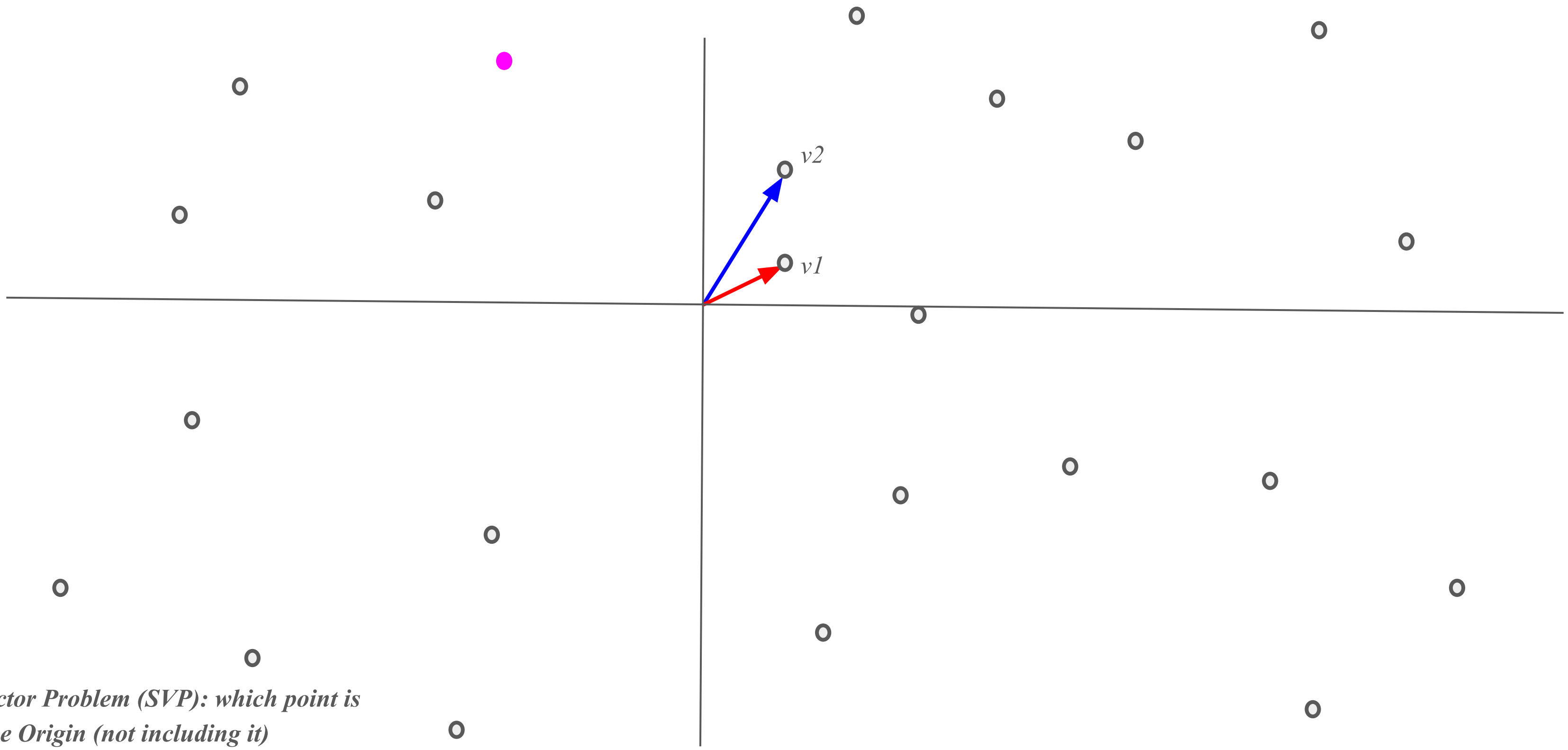


V1 + 2v2



-V1 + 2v2

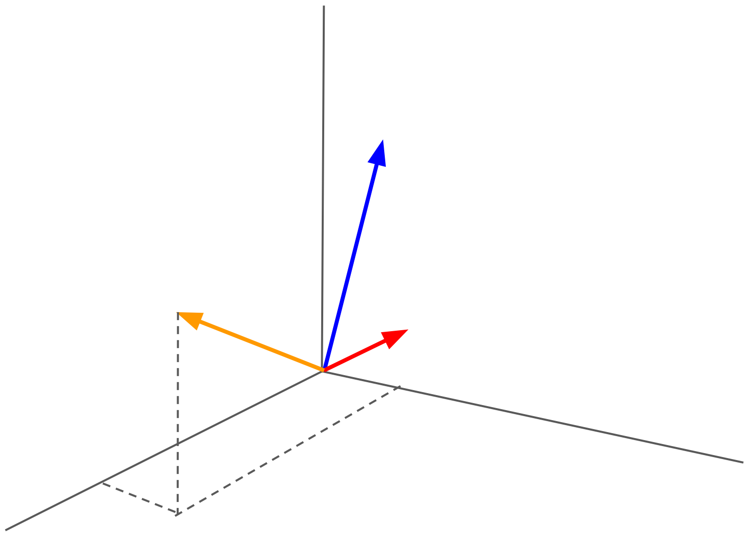
Lattice based cryptography



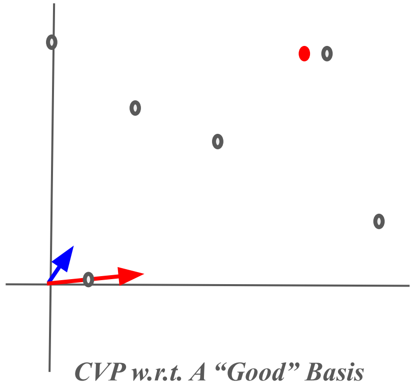
Shortest Vector Problem (SVP): which point is closest to the Origin (not including it)
Closest Vector Problem (CVP): which point is closest to a given point (not including it)

A Two-Dimensional Lattice: a.V1 + b.V2; a, b in Base-Field

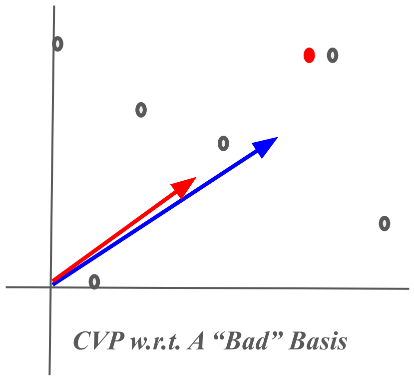
Lattice based cryptography



A Three-Dimensional Lattice with three three-dimensional Vectors V1, V2 and V3
In General : n-Dimensional Lattice with n n-dimensional Vectors V1, …, Vn



CVP w.r.t. A "Good" Basis



CVP w.r.t. A "Bad" Basis

GGH Encryption Scheme is designed based on Good and Bad Basis
Goldreich, Goldwasser, and Halevi

Lattice based cryptography

- **GGH Encryption Scheme is designed based on Good and Bad Basis**
- **GGH is not fully secure**

- **More Secure Approaches:-**
    - **Learning with Errors (LWE) Problem**
        - **Given a set of noisy linear equations, it is hard to determine the original secret values**
    - **Ring LWE Problem**
        - **more efficient variant of LWE used in practical cryptographic schemes**

Lattice based cryptography

Learning with Errors (LWE) Problem

Learning Without Error

<s1 = 10, s2 = 82, s3 = 50, s4 = 5>

Secret Vector

a1.s1+ a2. s2+ a3. s3+ a4.s4 ≡ c mod p;   a1, a2, a3, a4 ∈ Zp

77.s1 + 7.s2 + 28.s3 + 23.s4 ≡ 11 (mod 89)
21.s1 + 19.s2 + 30.s3 + 48.s4 ≡ 37 (mod 89)
4.s1 + 24.s2 + 33.s3 + 38.s4 ≡ 21 (mod 89)
8.s1 + 20.s2 + 84.s3 + 61.s4 ≡ 84 (mod 89)

Anyone Can Calculate the Secret Vector from the above Set of Equations

Equations With Error

77.s1 + 7.s2 + 28.s3 + 23.s4 ≡ 11 + 2 ≡ 13 (mod 89)
21.s1 + 19.s2 + 30.s3 + 48.s4 ≡ 37 - 1 ≡ 36 (mod 89)
4.s1 + 24.s2 + 33.s3 + 38.s4 ≡ 21 - 2 ≡ 19 (mod 89)
8.s1 + 20.s2 + 84.s3 + 61.s4 ≡ 84 + 2 ≡ 86 (mod 89)

Hard to find the Secret Vector from the above Set of Equations

Secret Vector: Private-Key

Set of Equns with Error: Public-Key

Lattice based cryptography

Learning with Errors (LWE) Problem

Encryption

Public-Key

77.s1 + 7.s2 + 28.s3 + 23.s4 ≡ 11 + 2 ≡ 13 (mod 89)
21.s1 + 19.s2 + 30.s3 + 48.s4 ≡ 37 - 1 ≡ 36 (mod 89)
4.s1 + 24.s2 + 33.s3 + 38.s4 ≡ 21 - 2 ≡ 19 (mod 89)
8.s1 + 20.s2 + 84.s3 + 61.s4 ≡ 84 + 2 ≡ 86 (mod 89)

Sum of the Above Eqns.:-

21.s1 + 70.s2 + 86.s3 + 81.s4 ≡ 64 + 1 ≡ 65 (mod 89)

Encryption of bit '0':-
21.s1 + 70.s2 + 86.s3 + 81.s4 ≡ 65 (mod 89)

Encryption of bit '1':-
21.s1 + 70.s2 + 86.s3 + 81.s4 ≡ 65 + 44 = 20 (mod 89)

Decryption

Private-Key: Secret Vector

<s1 = 10, s2 = 82, s3 = 50, s4 = 5>

Step-1: Plug-in the values in the Eqn.:-

*21.10 + 70.82 + 86.50 + 81.5 ≡ **64** (mod 89)*

Step-2: Subtract the RHS from the Received Value:-

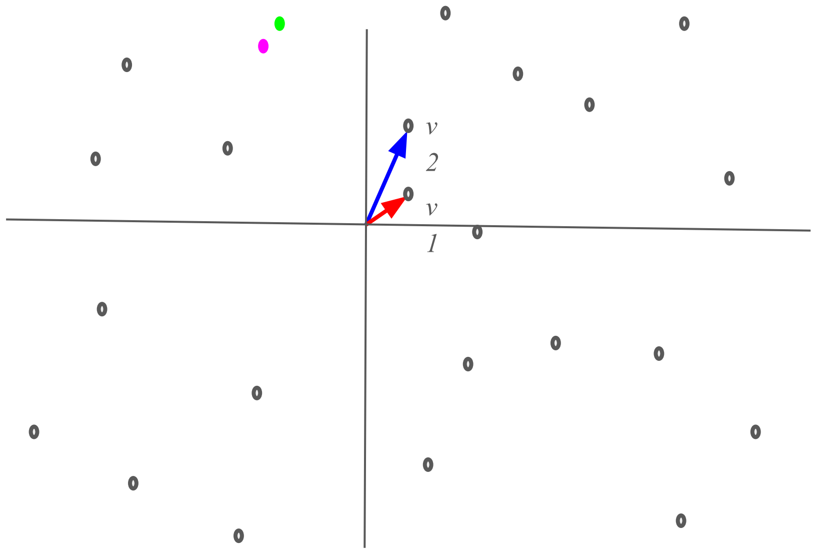
*65 - 64 ≡ **1** (mod 89) : Close to ZERO ===> **Interpreted as '0'***
*65 - 20 ≡ **45** (mod 89): Faraway from ZERO ===> **Interpreted as '1'***

Lattice based cryptography

Learning with Errors (LWE) Problem

Combined with Integer Lattice Problem

- RHS of the equations are encoded as a lattice with small error
- Obtaining the correct equation reduces to solving closest vector problem



Lattice based cryptography

Constructions Using Lattice Based Cryptography:-

- **Public-key encryption (e.g., Kyber, NTRU)**
- **Digital signatures (e.g., Dilithium, Falcon)**
- **Fully homomorphic encryption (FHE)**
- **Zero-knowledge proofs**
- **Identity-based encryption**

Challenges:-

- **Larger key sizes compared to RSA/ECC**
- **Implementation complexity in hardware and software**

NIST Standards:-

- **Kyber (encryption)**
- **Dilithium (signatures)**

Code-based cryptography

Error correction codes

- Digital media is exposed to memory corruption.
- Many systems check whether data was corrupted in transit:
  - ISBN numbers have check digit to detect corruption.
  - ECC RAM detects up to two errors and can correct one error.
    64 bits are stored as 72 bits: extra 8 bits for checks and recovery.
- In general, k bits of data get stored in n bits, adding some redundancy.
- If no error occurred, these n bits satisfy $n - k$ parity check equations; else can correct errors from the error pattern.
- Good codes can correct many errors without blowing up storage too much;
  offer guarantee to correct t errors (often can correct or at least detect more).

Example: Hamming Code

Slides from Tanja Lange,Tung Chou and Christiane Peters

Code-based cryptography

Hamming Code?

- Linear error-correcting code
- Developed by Richard Hamming in 1950
- Detects & corrects single-bit errors.
- Adds parity bits at power-of-2 positions
  - Parity bits at positions 1, 2, 4 (powers of 2).
  - Data bits fill remaining positions.
  - For (7,4): [P1 P2 D1 P3 D2 D3 D4]

Code-based cryptography

An Example :-

- **Data bits: 1 0 1 1** (D1 D2 D3 D4).
- Codeword layout: [P1 P2 D1 P3 D2 D3 D4].

Parity Coverage:-

- P1 covers bits 1,3,5,7
- P2 covers bits 2,3,6,7
- P3 covers bits 4,5,6,7
- These sets define the parity check conditions

Positions: 1 2 3 4 5 6 7
1: 0 0 1
2: 0 1 0
3: 0 1 1
4: 1 0 0
5: 1 0 1
6: 1 1 0
7: 1 1 1

Calculate Parity Bits:-

Bit Position: [**1 2 3 4 5 6 7**]

- Data bits placed: [_ _ 1 _ 0 1 1]
- **P1:** bits 1,3,5,7 → ? 1 0 1 → sum=2 → P1=0
- **P2:** bits 2,3,6,7 → ? 1 1 1 → sum=3 → P2=1
- P3: bits 4,5,6,7 → ? 0 1 1 → sum=2 → P4=0
- **Final: [0 1 1 0 0 1 1]**.

Code-based cryptography

Generator Matrix (G)

- G (7,4) = [I | P]:

  [1 0 0 0 | 1 1 0]

  [0 1 0 0 | 1 0 1]

  [0 0 1 0 | 1 0 0]

  [0 0 0 1 | 0 1 1]
- Encode: codeword = data × G.

Parity Check Matrix (H)

- H = [P^t | I]:

  [1 1 1 0 | 1 0 0]

  [1 0 0 1 | 0 1 0]

  [0 1 0 1 | 0 0 1]
- Check: **syndrome** = H × $codeword^t$.

Error Detection Example

- Received: 0 1 1 0 1 1 1 (bit 5 flipped)
- **Syndrome** = H × $received^t$ → gives non-zero.
- Result tells position of the error bit.
- Example: syndrome = 101 → binary 5 → bit 5 is wrong.

Linear codes

A binary linear code C of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_2^n$.
C is usually specified as

- the row space of a generating matrix $G \in \mathbb{F}_2^{k \times n}$

$$C = \{\mathbf{m}G | \mathbf{m} \in \mathbb{F}_2^k\}$$

- the kernel space of a parity-check matrix $H \in \mathbb{F}_2^{(n-k) \times n}$

$$C = \{\mathbf{c} | H\mathbf{c}^\intercal = 0, \ \mathbf{c} \in \mathbb{F}_2^n\}$$

  Leaving out the $^\intercal$ from now on.
- Names: code word $\mathbf{c}$, error vector $\mathbf{e}$, received word $\mathbf{b} = \mathbf{c} + \mathbf{e}$.

Example: Hamming code

Parity check matrix ($n = 7, k = 4$):

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

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{cccccccc} b_0 & +b_1 & & +b_3 & +b_4 & & & = & 0 \\ b_0 & & +b_2 & +b_3 & & +b_5 & & = & 0 \\ & b_1 & +b_2 & +b_3 & & & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., $1, 0, 1$ means

Example: Hamming code

Parity check matrix ($n = 7, k = 4$):

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

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{cccccccccc} b_0 & +b_1 & & +b_3 & +b_4 & & & = & 0 \\ b_0 & & +b_2 & +b_3 & & +b_5 & & = & 0 \\ & b_1 & +b_2 & +b_3 & & & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., $1, 0, 1$ means b_1 flipped.

Example: Hamming code

Parity check matrix ($n = 7, k = 4$):

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

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{ccccccccc} b_0 & +b_1 & & +b_3 & +b_4 & & & = & 0 \\ b_0 & & +b_2 & +b_3 & & +b_5 & & = & 0 \\ & b_1 & +b_2 & +b_3 & & & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., $1, 0, 1$ means b_1 flipped.
In math notation, the failure pattern is $H \cdot \mathbf{b}$.

Linear codes are linear

Example with generator matrix:

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

$\mathbf{c} = (111)G = (10011)$ is a code word.

Linear codes are linear

Example with generator matrix:

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

$\mathbf{c} = (111)G = (10011)$ is a code word.

Linear codes are linear:
The sum of two code words is a code word:

Linear codes are linear

Example with generator matrix:

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

$\mathbf{c} = (111)G = (10011)$ is a code word.

Linear codes are linear:
The sum of two code words is a code word:

$$\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{m}_1 G + \mathbf{m}_2 G = (\mathbf{m}_1 + \mathbf{m}_2)G.$$

Same with parity-check matrix:

Linear codes are linear

Example with generator matrix:

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

$\mathbf{c} = (111)G = (10011)$ is a code word.

Linear codes are linear:
The sum of two code words is a code word:

$$\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{m}_1 G + \mathbf{m}_2 G = (\mathbf{m}_1 + \mathbf{m}_2)G.$$

Same with parity-check matrix:

$$H(\mathbf{c}_1 + \mathbf{c}_2) = H\mathbf{c}_1 + H\mathbf{c}_2 = 0 + 0 = 0.$$

Hamming weight and distance

- The Hamming weight of a word is the number of nonzero coordinates.

$$\mathrm{wt}(1,0,0,1,1) = 3$$

- The Hamming distance between two words in $\mathbb{F}_2^n$ is the number of coordinates in which they differ.

$$d((1,1,0,1,1),(1,0,0,1,1)) =$$

Hamming weight and distance

- The Hamming weight of a word is the number of nonzero coordinates.

$$\mathrm{wt}(1,0,0,1,1) = 3$$

- The Hamming distance between two words in $\mathbb{F}_2^n$ is the number of coordinates in which they differ.

$$d((1,1,0,1,1),(1,0,0,1,1)) = 1$$

Hamming weight and distance

- The Hamming weight of a word is the number of nonzero coordinates.

$$\mathrm{wt}(1,0,0,1,1) = 3$$

- The Hamming distance between two words in $\mathbb{F}_2^n$ is the number of coordinates in which they differ.

$$d((1,1,0,1,1),(1,0,0,1,1)) = 1$$

  The Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ equals the Hamming weight of $\mathbf{x}+\mathbf{y}$:

$$d((1,1,0,1,1),(1,0,0,1,1)) = \mathrm{wt}(0,1,0,0,0).$$

Minimum distance

- The minimum distance of a linear code C is the smallest Hamming weight of a nonzero code word in C.

$$d = \min_{0 \neq \mathbf{c} \in C} \{\mathrm{wt}(\mathbf{c})\} = \min_{\mathbf{b} \neq \mathbf{c} \in C} \{d(\mathbf{b}, \mathbf{c})\}$$

- In code with minimum distance $d = 2t + 1$, any vector $\mathbf{x} = \mathbf{c} + \mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) \leq t$ is uniquely decodable to $\mathbf{c}$;
  i. e. there is no closer code word.

Decoding problem

Decoding problem: find the closest code word $\mathbf{c} \in C$ to a given $\mathbf{x} \in \mathbb{F}_2^n$, assuming that there is a unique closest code word. Let $\mathbf{x} = \mathbf{c} + \mathbf{e}$. Note that finding $\mathbf{e}$ is an equivalent problem.

- If $\mathbf{c}$ is t errors away from $\mathbf{x}$, i.e., the Hamming weight of $\mathbf{e}$ is t, this is called a t-error correcting problem.
- There are lots of code families with fast decoding algorithms, e.g., Reed–Solomon codes, Goppa codes/alternant codes, etc.
- However, the general decoding problem is hard: Information-set decoding (see later) takes exponential time.

The McEliece cryptosystem I

- Due to Robert McEliece 1978.
- Let C be a length-n binary Goppa code Γ of dimension k with minimum distance $2t+1$ where $t \approx (n-k)/\log_2(n)$; original parameters (1978) $n = 1024$, $k = 524$, $t = 50$.
- The McEliece secret key consists of a generator matrix G for Γ, an efficient t-error correcting decoding algorithm for Γ; an $n \times n$ permutation matrix P and a nonsingular $k \times k$ matrix S.
- n, k, t are public; but Γ, P, S are randomly generated secrets.
- The McEliece public key is the $k \times n$ matrix $G' = SGP$.

The McEliece cryptosystem II

- Encrypt: Compute $\mathbf{m}G'$ and add a random error vector $\mathbf{e}$ of weight t and length n. Send $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
- Decrypt: Compute $\mathbf{y}P^{-1} = \mathbf{m}G'P^{-1} + \mathbf{e}P^{-1} = (\mathbf{m}S)G + \mathbf{e}P^{-1}$. This works because $\mathbf{e}P^{-1}$ has the same weight as $\mathbf{e}$

The McEliece cryptosystem II

- Encrypt: Compute $\mathbf{m}G'$ and add a random error vector $\mathbf{e}$ of weight t and length n. Send $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
- Decrypt: Compute $\mathbf{y}P^{-1} = \mathbf{m}G'P^{-1} + \mathbf{e}P^{-1} = (\mathbf{m}S)G + \mathbf{e}P^{-1}$.
  This works because $\mathbf{e}P^{-1}$ has the same weight as $\mathbf{e}$
  because P is a permutation matrix.
  Use fast decoding to find $\mathbf{m}S$ and $\mathbf{m}$.
- Attacker is faced with decoding $\mathbf{y}$ to nearest code word $\mathbf{m}G'$ in the code generated by G'.
  This is general decoding if G' does not expose any structure.

Hash-based cryptography

- Uses secure hash functions only
- Simple, conservative design
- Survives quantum attacks
- Proven: decades of research

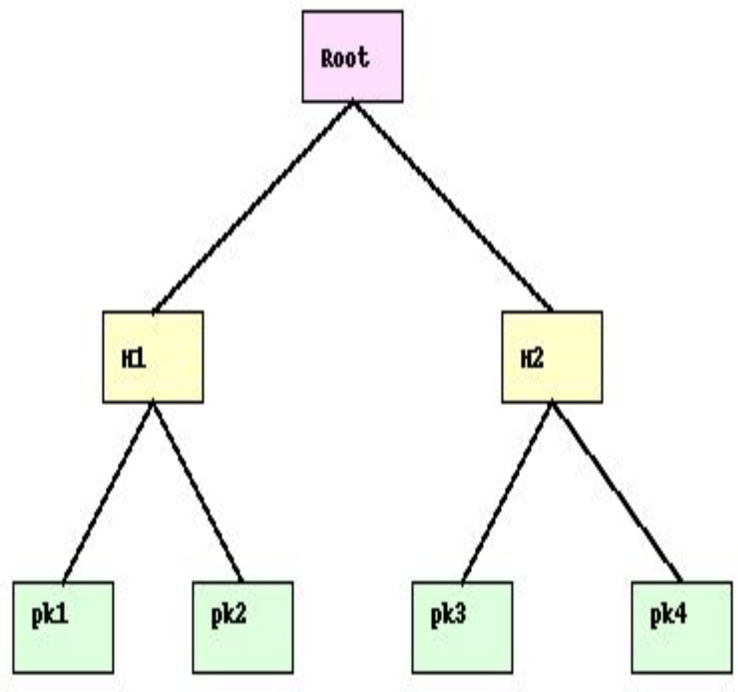
Lamport OTS

- Generate 2 secrets per message bit
- Sign by revealing secrets
- Verify: hash revealed secrets match public key
- Use once only!

Hash-based cryptography

Many-Time: Merkle Tree

- Combine many OTS keys
- Build Merkle tree → Root = master public key
- Sign: OTS + Merkle proof
- Verify: OTS signature + proof to root



Real-World Use: XMSS

- XMSS = eXtended Merkle Signature Scheme (RFC 8391)
- Used for secure IoT firmware updates
- Small public key, huge number of signatures
- Very strong quantum resistance

Quantum Random Oracle Model

What is a Random Oracle?

In classical cryptography, a Random Oracle is an idealized hash function:

- It’s a theoretical black box that responds to every unique query with a truly random output.
- If you query it with the same input again, it gives the same output.
- No algorithm can predict the output except by asking the oracle.

In practice, cryptographic hash functions (like SHA-256) are often modeled as random oracles in security proofs to simplify analysis.

Quantum Random Oracle Model

What is the Quantum Random Oracle Model (QROM)?

In the Quantum Random Oracle Model, the adversary is assumed to have quantum capabilities. That means:

- The adversary can make quantum queries to the random oracle.
- They can query in superposition: instead of asking the oracle for H(x) for a single x, they can prepare a quantum state that's a superposition of multiple inputs, and the oracle must respond coherently to that entire state.

Quantum Random Oracle Model

Why is QROM needed?

Modern hash-based proofs rely heavily on the assumption that an attacker can only make classical queries. However, if someone has a quantum computer:

- They can run Grover’s algorithm to get a quadratic speedup for finding pre-images.
- They can use quantum queries to break protocols that are secure in the classical random oracle model but fail when queries are made in superposition.

QROM is a stronger, more realistic model for analyzing post-quantum cryptography.

Quantum Random Oracle Model

What changes in proofs?

Proving security in the QROM is trickier:

- The simulator must answer all possible quantum queries consistently — which is technically challenging.
- Classical “rewinding” techniques for proofs don’t work directly on quantum adversaries.

New proof techniques like measure-and-reprogram, semi-classical oracles, or special quantum rewinding tricks are used instead.

Conclusions & Future Research Challenges

- **PQC algorithms generally require larger key sizes and more complex computations compared to traditional cryptographic methods.** This can lead to higher processing power and memory requirements, affecting performance, especially in resource-constrained environments such as Internet of Things (IoT) devices and real-time systems.

- **Many enterprises lack the necessary knowledge and expertise to implement PQC solutions effectively.** The complexity of these new cryptographic methods requires security professionals to obtain specialized training and increase their ability to adapt.

- **Unlike traditional encryption algorithms that have been standardized and widely adopted for decades, PQC is still evolving, and many existing systems are not designed to handle post-quantum cryptographic primitives. Implementing PQC requires rewriting cryptographic libraries, updating protocols, and ensuring backward compatibility, all of which introduce potential vulnerabilities and security risk.**

- **Despite the threat of quantum computing quickly approaching, many organizations are occupied with other priorities, such as adapting to AI and other new technologies, which inevitably will lead to limited engagement with quantum computing and its security implications.**

- **While NIST has made significant progress, the landscape is still evolving. There is still a lack of comprehensive guidance and uncertainty regarding the appropriate algorithms to choose.**

Ref: "Post-Quantum Cryptography: A Call to Action", by Collin Beder, CSX-P, CET, Security+ (www.isaca.org)

References

1 Nayak, C.; "Microsoft Unveils Majorana 1, the World's First Quantum Processor Powered by Topological Qubits," Microsoft, 19 February 2025; Conover, E.; "The New Light-Based Quantum Computer Jiuzhang has Achieved Quantum Supremacy," Science News, 3 December 2020; Lee, J.; "IBM Launches its Most Powerful Quantum Computer With 433 Qubits," Reuters, 9 November 2022; Newman, M.; Satzinger, K.; et al.; "Making Quantum Error Correction Work," Google Research, 9 December 2024
2 Keyfactor, "Harvest Now, Decrypt Later: A New Form of Attack," 29 April 2024
3 ISACA® Pulse Poll on Quantum Computing, 2025
4 Bolgar, C.; "Microsoft's Majorana 1 Chip Carves New Path for Quantum Computing," Microsoft, 19 February 2025
5 Farlini, E.; "Scientists Question Microsoft's Quantum Computing 'Breakthrough,'" PC Mag, 10 March 2025
6 National Institute of Standards and Technology (NIST), NIST Internal Report NIST IR 8545 Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process, USA, March 2025
7 Wickramasinghe, S.; "RSA Algorithm in Cryptography: Rivest Shamir Adleman Explained," Splunk Blogs, 26 November 2024
8 GeeksforGeeks, "Digital Signature Algorithm (DSA)," 13 February 2025
9 GeeksforGeeks, "Blockchain – Elliptic Curve Digital Signature Algorithm (ECDSA)," 29 November 2022
10 Gillis, A.; "Diffie-Hellman Key Exchange (Exponential Key Exchange)," TechTarget
11 Relyea, R.; "Post-Quantum Cryptography: Lattice-Based Cryptography," Red Hat Blog, 30 October 2023
12 Relyea; "Post-Quantum Cryptography: Lattice-Based Cryptography"
13 NIST, "FIPS 203 Module-Lattice-Based Key-Encapsulation Mechanism Standard," 13 August 2024
14 Cryptographic Suite for Algebraic Lattices, "CRYSTALS"
15 NIST, "FIPS 204 Module-Lattice-Based Digital Signature Standard," 13 August 2024
16 Cryptographic Suite for Algebraic Lattices, "CRYSTALS"
17 Relyea, R.; "Post-Quantum Cryptography: Hash-Based Signatures," 27 October 2022
18 NIST, "FIPS 205 Stateless Hash-Based Digital Signature Standard," 13 August 2024
19 Yevchenko, A.; "What Exactly is Grover's Algorithm?," Medium, 9 December 2021
20 NIST, "FIPS 205 Stateless Hash-Based Digital Signature Standard"
21 NIST, "FIPS 205 Stateless Hash-Based Digital Signature Standard"
22 Stateless Hash-based Signatures, "SPHINCS+"
23 Anglen, J.; "Future-Proofing Blockchain: Embracing Quantum-Resistant Cybersecurity in 2024," Rapid Innovation
24 Anglen; "Future-Proofing Blockchain: Embracing Quantum-Resistant Cybersecurity in 2024"
25 Anglen; "Future-Proofing Blockchain: Embracing Quantum-Resistant Cybersecurity in 2024"

References

25 Anglen; "Future-Proofing Blockchain: Embracing Quantum-Resistant Cybersecurity in 2024"
26 Anglen; "Future-Proofing Blockchain: Embracing Quantum-Resistant Cybersecurity in 2024"
27 Gaborit, P.; Deneuville, J.C.; Hamming Quasi-Cyclic (HQC), 19 February 2025
28 Boutin, C.; "NIST Selects HQC as Fifth Algorithm for Post-Quantum Encryption," NIST News, 11 March 2025
29 Lokhande, B.; "Post-Quantum Cryptography for Internet and WebPKI: Where are We Now and How Do You Prepare?," Redshift Blog, 20 February 2025
30 Ivezic, M.; "Post-Quantum Cryptography PQC Challenges," Post Quantum, 1 June 2023
31 IDEMIA, "Key Obstacles to Post-Quantum Cryptography (PQC) Adoption," 26 February 2025
32 Celi, S.; Sullivan, N.; "The Post-Quantum Future: Challenges and Opportunities," The Cloudfare Blog, 25 February 2025
33 Bos, J.; Cloostermans, C.; et al.; Post-Quantum Cryptographic Migration Challenges for Embedded Devices, NXP
34 Bousquette, I.; "Quantum Computing Is Closer Than Ever. Everybody's Too Busy to Pay Attention," The Wall Street Journal, 13 February 2025
35 NIST, "Post-Quantum Cryptography (PQC)" 27 March 2025
36 CISA, "CISA, NSA and NIST Publish New Resource for Migrating to Post-Quantum Cryptography," CISA, 21 August 2023
37 QED-C, "The Quantum Consortium" 2025
38 Harishankar, R.; Osborne, M.; et al.; "Crypto-Agility and Quantum-Safe Readiness," IBM, 19 June 2024

https://www.microsoft.com/en-us/research/project/post-quantum-cryptography/
https://openquantumsafe.org/
https://github.com/open-quantum-safe/liboqs
https://www.redhat.com/en/blog/post-quantum-cryptography-lattice-based-cryptography

THANK YOU