Quantum Cryptography

Quantum Cryptography is the science of using quantum mechanical properties to perform cryptographic tasks. Uses photons to transmit data, the photons represent binary bits. The properties of quantum mechanics that allow for cryptography are

- Particles can exist in more than one place or state at a time
- A quantum property cannot be observed without changing or disturbing it
- Whole particles cannot be copied.
- Photons are generated randomly in one of two quantum states

These make it impossible to measure the quantum state of the system without changing it.

Quantum Key Distribution (QKD)

Quantum Key Distribution is a cryptographic method that uses the principles of quantum mechanics to securely share encryption keys between two parties. Unlike classical cryptographic systems, which rely on the assumed hardness of mathematical problems, QKD offers **unconditional security** rooted in the physical laws of nature.

Key Concepts

1. Heisenberg's Uncertainty Principle
Any attempt to measure or observe a quantum state will disturb it. This principle ensures that if an eavesdropper tries to intercept the communication, they will leave detectable traces.

2. No-Cloning Theorem
It is impossible to perfectly copy an unknown quantum state. This prevents an attacker from duplicating the quantum information to steal the key undetected.

Protocol

Step 1: Transmission
Alice prepares and sends photons (quantum bits or qubits) in one of two bases: rectilinear (+) or diagonal (×), with each photon representing a 0 or 1.

Step 2: Measurement
Bob randomly chooses a basis (either + or ×) to measure each photon he receives. Due to quantum mechanics, if he chooses the correct basis, he gets the correct bit. Otherwise, the result is random.

Step 3: Basis Reconciliation
Alice and Bob communicate over a classical channel (public, but authenticated) to compare the bases used—not the actual bit values. They discard bits where their bases didn't match.

Step 4: Key Sifting
The remaining bits form the raw key.

Step 5: Error Correction and Privacy Amplification
They perform error correction to remove noise or measurement errors and apply privacy amplification to eliminate any information potentially known by an eavesdropper.

How Quantum Key Distribution Prevents Eavesdropping

1. **Alice sends Bob a stream of photons**, each polarized to represent a 0 or 1 using one of two possible polarizer types.

2. **Bob randomly chooses a polarizer** to measure each photon. He doesn't know which one Alice used, so he guesses.

3. **Eve tries to intercept the photons**, using the same tools as Bob. She also guesses the polarizer type to measure the photons.

4. **If Eve guesses wrong**, her measurement alters the photon's quantum state.

5. **Eve sends the altered photon to Bob**, who then measures it. If the photon was disturbed, Bob might get the wrong result.

6. **Later, Alice and Bob compare which polarizers they used**, and only keep results where they matched.

7. **They publicly compare a portion of the remaining bits** to check for errors. If too many discrepancies appear, they know someone was listening.

8. **They discard the compromised key** and repeat the process to generate a new, secure key.

Advantages of Quantum Cryptography:

- **Unconditional security**: Based on laws of quantum mechanics, not computational assumptions.

- **Eavesdropping detection**: Any interception alters quantum states, making it detectable.
- **Future-proof**: Resistant to attacks from quantum computers, unlike classical cryptography.

Limitations:

- **Distance constraints**: Photon loss in fiber limits range; long-distance requires satellites or quantum repeaters.
- **High cost and complexity**: Requires specialized hardware (photon sources, detectors).
- **Low key generation rate**: Slower compared to classical key distribution methods.
- **Integration challenges**: Difficult to combine with existing classical infrastructure.

Applications of Quantum Cryptography:

- **Military and government communication**: High-security transmissions.
- **Financial sector**: Securing interbank transfers and transactions.
- **Healthcare**: Protecting sensitive patient data.
- **Critical infrastructure**: Securing power grids, defense systems, and communications.
- **Quantum key distribution**: The best-known and developed application of quantum cryptography is QKD
- **Mistrustful quantum cryptography**: the participating parties do not trust each other but with QC, they make sure that both have not cheated

What is Homomorphic Encryption?

Homomorphic Encryption (HE) is a form of encryption that allows computations to be performed directly on **encrypted data** (ciphertext), without needing to decrypt it first. The result, when decrypted, matches the result of operations performed on the original plaintext.

For example:
Encrypt(A) + Encrypt(B) = Encrypt(A + B)

This enables secure data processing while preserving privacy.

Types of Homomorphic Encryption:

1. **Additive Homomorphic Encryption**
   Supports addition operations on ciphertexts.
   Example:
   Encrypt(a) + Encrypt(b) = Encrypt(a + b)
   Used in: Voting systems, private data aggregation
   Scheme: Paillier encryption

2. **Multiplicative Homomorphic Encryption**
   Supports multiplication operations on ciphertexts.
   Example:
   Encrypt(a) × Encrypt(b) = Encrypt(a × b)
   Used in: Secure auctions, financial calculations
   Scheme: RSA (partially)

3. **Fully Homomorphic Encryption (FHE)**
   Supports **both** addition and multiplication on encrypted data, enabling arbitrary computations.
   Used in: Secure cloud computing, encrypted machine learning
   Limitation: Very slow and resource-intensive

Advantages:

- **Data privacy during computation**: Sensitive data can be processed without being exposed.

- **Ideal for cloud computing**: Enables outsourcing of computation to untrusted servers while keeping data encrypted.
- **Supports secure multi-party computations**: Different parties can compute on shared encrypted data without revealing their inputs.
- **Regulatory compliance**: Helps meet data protection laws (e.g., GDPR, HIPAA) by ensuring data stays encrypted.

Limitations:

- **Performance overhead**: Operations on encrypted data are significantly slower than on plaintext.
- **Limited operation support**: Some schemes only allow specific types of operations (e.g., addition or multiplication).
- **Complex implementation**: Requires careful cryptographic design and more computational resources.
- **Large ciphertext sizes**: Encrypted data can be much larger than plaintext, increasing storage and transmission costs.

Applications:

- **Cloud computing and storage**: Clients can store and compute on encrypted data securely.
- **Healthcare**: Enables analysis on encrypted patient data without privacy breaches.
- **Finance**: Allows banks to perform calculations (e.g., risk analysis) on encrypted customer data.
- **Machine learning on encrypted data**: Securely train or evaluate models on confidential datasets.
- **Electronic voting**: Ensures vote privacy while still allowing correct tallying.

Example Usecase

Homomorphic encryption allows hospitals to store and process encrypted Electronic Health Records (EHRs) on the cloud without exposing sensitive patient data. Healthcare providers can perform operations like data analysis or diagnostics directly on encrypted records, and only the final results are decrypted locally. This ensures patient privacy, complies with regulations like HIPAA, and enables secure use of cloud-based services without risking data breaches.

Secure Multi Party Communication

What is Secure Multi-Party Computation (SMPC)?

SMPC allows multiple parties to jointly compute a function over their private inputs **without revealing those inputs to each other**. Each party only learns the final result, not anyone else's data.

Advantages:

- **Data privacy preserved**: No party learns others' inputs.
- **Collaborative analytics**: Enables joint computation across organizations without sharing sensitive data.
- **No trusted third party required**: Security is maintained without needing a central authority.
- **Resilience to breaches**: Even if one party is compromised, individual data remains protected.

Disadvantages:

- **High computational overhead**: Slower than normal computation due to encryption and data splitting.
- **Complex implementation**: Requires careful design and secure protocols.
- **Limited scalability**: Performance degrades with more parties or large datasets.
- **Communication cost**: Requires multiple rounds of secure communication between parties.

Applications:

- **Collaborative medical research**: Hospitals can analyze combined patient data without exposing private records.
- **Financial risk analysis**: Banks compute shared risks without revealing internal data.
- **Secure bidding/auctions**: Participants can submit bids without revealing values to competitors.
- **Privacy-preserving machine learning**: Train models on distributed, sensitive datasets (e.g., across hospitals or banks).

Zero Knowledge Proofs

What are Zero-Knowledge Proofs?

A **Zero-Knowledge Proof (ZKP)** is a cryptographic technique that allows one party (called the *prover*) to convince another party (the *verifier*) that a given statement is true, **without revealing any additional information** apart from the fact that the statement is indeed true.

The central idea is: *"I can prove to you that I know something, without revealing what that something is."*

How It Works (Conceptually):

ZKPs rely on three main properties:

1. **Completeness**:
   If the statement is true, an honest prover can convince an honest verifier.

2. **Soundness**:
   If the statement is false, no cheating prover can convince the verifier, except with some tiny probability.

3. **Zero-Knowledge**:
   The verifier learns nothing beyond the fact that the statement is true — no actual information about the secret is revealed.

Example (Intuitive Analogy):

A common analogy is the **"Where's Waldo?" proof**:

- Suppose you have found Waldo in a puzzle book, but you don't want to reveal where he is.

- You cover the entire page with a large sheet except for a small cutout showing just Waldo.

- This proves you know his location, but you haven't shown the verifier anything else on the page.

This is the essence of zero-knowledge: **proof without disclosure**.

Advantages	Disadvantages
Simplicity Does not require any complicated encryption methods.	**Limited** The protocols for ZKP's usually rely on mathematical equations and numerical answers. Any other method requires a translation.
Privacy Increases the privacy of users by avoiding the reveal of personal information in public blockchains.	**Requires a large amount of computing power** There are around 2000 computations per ZKP transaction that each require a certain amount of time to process.
Security Strengthens security of information by replacing ineffective authentication methods.	**Restricted** If the originator of a transaction forgets their information, all the data associated with it is lost.
Scalability Increases blockchain throughput and scalability.	**Vulnerability** Potential vulnerability to advanced technologies like quantum computing.

Applications:

- **Cryptocurrencies**:
    - *Zcash* uses zk-SNARKs to enable private transactions (hiding sender, receiver, and amount).
- **Authentication**:
    - Prove identity or age without revealing personal details.
- **Blockchain smart contracts**:

    - Prove something happened off-chain without putting data on-chain (e.g., proving a vote, a payment, or a credential).

- **E-voting systems**:
    - Ensure that votes are valid without disclosing individual choices.

- **Access control and privacy**:
    - For example, proving you're in a geographic location or a member of a group without revealing your identity.

Cryptographic Obfuscation

What is Cryptographic Obfuscation?

Obfuscation is the process of transforming a program into a version that is **functionally identical** to the original but **unintelligible** or **hard to analyze**, even when an attacker has full access to it. In cryptographic contexts, it aims to protect the **logic, data, and control flow** of the code to enhance software and data security.

Security Goals Achieved:

1. **Confidentiality**
   Ensures that sensitive parts of a program remain hidden from unauthorized users.

2. **Intellectual Property Protection**
   Prevents reverse engineering of proprietary logic or algorithms embedded in the software.

3. **Access Control Enforcement**
   Even if attackers gain access to the program binary, they can't extract useful or exploitable information.

4. **Privacy-Preserving Computation**
   Useful in applications like secure Multi-Party Computation (MPC) and functional encryption to conceal sensitive data and logic.

Common Techniques Used in Obfuscation (Any 5):

1. **Renaming**
   Variable and method names are altered to meaningless or invisible strings, removing semantic clues from the code.

2. **Packing**
   The program is compressed or encoded into a form that conceals its structure, delaying

reverse engineering.

3. **Control Flow Obfuscation**
   Logical flow is tangled into unstructured "spaghetti code" that hides the original program structure.

4. **Instruction Pattern Transformation**
   Common compiler instructions are replaced with equivalent, obscure variants to hinder pattern-based analysis.

5. **Dummy Code Insertion**
   Inserts non-functional, misleading code segments that do not affect behavior but confuse readers and tools.

6. **Metadata or Unused Code Removal**
   Removes comments, debug symbols, and dead code that could assist attackers in understanding or tracing the program.

7. **Opaque Predicate Insertion**
   Introduces always-true/false logical conditions that look complex but serve only to mislead analysts.

8. **Anti-Debugging**
   Incorporates checks and techniques that detect and resist debugging attempts, hindering step-by-step analysis.

9. **String Encryption**
   Obfuscates hardcoded strings by encrypting them and decrypting only at runtime, concealing useful keywords and paths.

10. **Code Transposition**
    Reorders code blocks or branches while preserving logic, making the flow harder to reconstruct.

Advantages:

- **Protects source code** from being easily analyzed or reverse engineered.

- **Enhances software security** by hiding algorithms, credentials, or embedded keys.

- **Useful in DRM** and mobile apps to safeguard licensing and in-app purchases.
- **Acts as a deterrent** against automated analysis and reverse engineering tools.

Disadvantages:

- **Performance Overhead**
  Some obfuscation methods can increase the program's runtime or size.
- **No Formal Guarantees**
  Obfuscation raises the difficulty of reverse engineering but doesn't make it impossible.
- **Maintenance Difficulty**
  Obfuscated code is extremely hard to debug, modify, or audit—even for the original developers.
- **Bypassable by Experts**
  Skilled reverse engineers and modern tools can still uncover the logic with enough effort.

Applications:

- **Software Protection**
  Prevents piracy and tampering by concealing core logic.
- **Mobile Apps and Games**
  Protects payment mechanisms, user data, and gameplay mechanics.
- **Cryptographic Implementations**
  Hides sensitive computations and cryptographic keys to protect against white-box attacks.
- **Digital Rights Management (DRM)**
  Used to enforce licensing and protect digital content.
- **Malware Evasion**
  Often used by malware developers to avoid detection by hiding intent and behavior.