What is Cryptography?

Cryptography is the science of securing information and communication using mathematical algorithms and codes to transform readable information (plaintext) into an unreadable format (ciphertext). Its main goals are confidentiality, integrity, authentication, and non-repudiation, ensuring only authorized access to data.

What is Cryptology?

Cryptology is the broader field that encompasses both cryptography (creating secure communication methods, encryption, signatures) and cryptanalysis (breaking/analyzing cryptographic methods to find weaknesses).

- **Cryptology = Cryptography + Cryptanalysis**
- **Cryptography** → designing secure systems (encryption, protocols).
- **Cryptanalysis** → breaking or attacking those systems.

- **Cipher** = তালার ডিজাইন/সিস্টেম (কীভাবে কাজ করে)
- **Key** = আপনার নির্দিষ্ট চাবি (যেটা দিয়ে খোলা যায়)

Symmetric vs Asymmetric Cryptosystems

Symmetric Cryptosystems (Secret-Key Cryptography)(CT*****)

Definition

Symmetric cryptography uses a **single shared secret key** for both encryption and decryption operations. Both the sender and receiver must possess the same key to communicate securely.

Key Characteristics

Key Usage:

- Same key for encryption and decryption
- The key must be kept secret between communicating parties
- Also called "secret-key" or "private-key" cryptography

Speed:

- Very fast and computationally efficient
- Can process large amounts of data quickly
- Suitable for bulk data encryption

Examples:

- **AES (Advanced Encryption Standard)**: Most widely used, supports 128, 192, or 256-bit keys
- **DES (Data Encryption Standard)**: Older standard, now considered insecure due to short 56-bit key
- **3DES (Triple DES)**: Enhanced version of DES
- **Blowfish, Twofish, ChaCha20**: Other popular algorithms

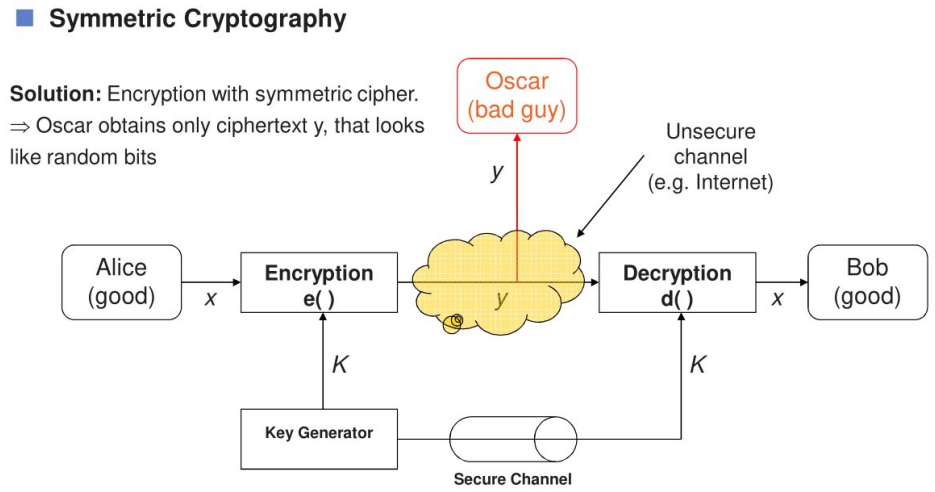
Advantages:

- High-speed encryption/decryption
- Low computational overhead
- Efficient for encrypting large volumes of data
- Simple implementation

Problems/Challenges:

- **Key distribution problem**: How to securely share the secret key between parties?
- **Key management**: With n users, you need n(n-1)/2 different keys for secure pairwise communication
- **No non-repudiation**: Cannot prove who sent the message
- **Scalability issues**: Managing keys becomes complex as the number of users increases

How Encryption Works



The second diagram shows how symmetric encryption protects their messages:

- Alice has a message (the **plaintext** x).
- She uses an **encryption function** $e(\cdot)$ with a secret **key** K to turn her message into **ciphertext** y.
- Oscar, listening on the channel, only sees y, not the original message.
- Bob uses the same secret key K and a **decryption function** $d(\cdot)$ to turn y back into the original message x.

Important terms:

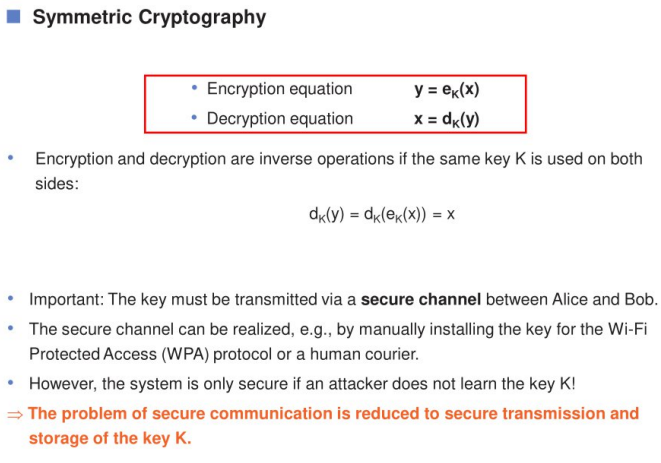
- **Plaintext (x)**: The readable message.
- **Ciphertext (y)**: The encrypted message, looks like random bits to outsiders.
- **Key (K)**: Shared secret used for both encryption and decryption.
- **Key space**: All possible keys $\{K_1, K_2, \ldots, K_n\}$.

Secure channel:

- Alice and Bob must share the key K over a secure method before communicating. Once they both have K, they can use it for encryption and decryption.

Summary check:

- Why does Oscar not learn anything useful? Because he only gets the ciphertext y, and without the key K, he can't decrypt it.



Asymmetric Cryptosystems (Public-Key Cryptography)

Definition

Asymmetric cryptography uses a **pair of mathematically related keys**: a public key (shared openly) and a private key (kept secret). What one key encrypts, only the other can decrypt.

Key Characteristics

Key Usage:

- **Public key**: Used for encryption, can be distributed freely to anyone
- **Private key**: Used for decryption, must be kept secret by the owner
- Two different but mathematically related keys

Speed:

- Significantly slower than symmetric encryption
- Computationally intensive operations
- Not suitable for encrypting large amounts of data directly

Examples:

- **RSA (Rivest-Shamir-Adleman)**: Most widely used, based on factoring large prime numbers
- **ECC (Elliptic Curve Cryptography)**: More efficient, provides same security with smaller keys
- **DSA (Digital Signature Algorithm)**: Primarily for digital signatures
- **Diffie-Hellman**: For key exchange
- **ElGamal**: For encryption and digital signatures

Advantages:

- **Solves key distribution problem**: Public keys can be shared openly
- **Better scalability**: Each user only needs one key pair (not n^2 keys)
- **Provides digital signatures**: Ensures authentication and non-repudiation
- **Supports key exchange**: Enables secure key establishment over insecure channels

Problems/Challenges:

- Much slower computation (100-1000x slower than symmetric)
- Requires more computational resources and power
- Not practical for encrypting large data volumes
- More complex mathematical operations

Hybrid Cryptosystems (Modern Approach)

Why Hybrid?

Modern systems combine both approaches to leverage their respective strengths while minimizing weaknesses.

How It Works:

1. **Asymmetric encryption** is used to securely exchange a symmetric session key
2. **Symmetric encryption** is then used to encrypt the actual data using that session key

Process Flow:

1. Generate random symmetric session key

2. Encrypt session key with recipient's public key (asymmetric)

3. Encrypt actual message with session key (symmetric)

4. Send both encrypted session key and encrypted message

5. Recipient decrypts session key with their private key (asymmetric)

6. Recipient decrypts message with session key (symmetric)

Real-World Examples:

- **HTTPS/TLS/SSL**: Web security protocols
- **PGP/GPG**: Email encryption
- **VPNs**: Virtual private networks
- **SSH**: Secure shell connections
- **Signal, WhatsApp**: Encrypted messaging apps

Benefits of Hybrid Approach:

- **Speed**: Fast symmetric encryption for data
- **Security**: Asymmetric encryption for key exchange
- **Scalability**: Easy key distribution
- **Efficiency**: Optimal resource utilization

Comparison Summary Table

Feature	Symmetric	Asymmetric
Keys	Same key for both operations	Public key (encrypt) + Private key (decrypt)
Speed	Very fast (suitable for large data)	Slower (100-1000x slower)
Key Length	Shorter keys (128-256 bits)	Longer keys (2048-4096 bits for RSA)
Examples	AES, DES, 3DES, ChaCha20	RSA, ECC, DSA

Best For	Bulk data encryption	Key exchange, digital signatures
Key Distribution	Difficult (secure channel needed)	Easy (public keys shared openly)
Scalability	Poor (n^2 key problem)	Good (n key pairs)
Computational Cost	Low	High
Use Cases	File encryption, disk encryption	Authentication, key exchange, signatures

Conclusion

Most modern cryptographic protocols use **hybrid schemes** that combine:

- **Symmetric ciphers** for efficient encryption and message authentication
- **Asymmetric ciphers** for secure key exchange and digital signatures

Classification of the Field of Cryptology(GPT)

Cryptology is the broader scientific field that encompasses both the creation and the breaking of secret codes. It is divided into two main branches:

1. Cryptography (Building Codes)

Definition: The art and science of designing secure communication systems, encryption algorithms, and security protocols to protect information.

Main Areas:

A. Symmetric Key Cryptography

- Uses the same secret key for encryption and decryption
- **Examples:** AES, DES, 3DES, ChaCha20, Blowfish
- **Applications:**
- Bulk data encryption

- Disk encryption
- Database encryption
- Secure file storage

B. Asymmetric Key Cryptography (Public-Key)

- Uses a pair of keys: public key and private key
- **Examples:** RSA, ECC (Elliptic Curve Cryptography), DSA
- **Applications:**
- Secure key exchange
- Digital certificates
- Email encryption (PGP/GPG)
- SSL/TLS for web security

C. Hash Functions

- One-way mathematical functions that convert data into fixed-size output
- Cannot be reversed to get original data
- **Properties:**
- Deterministic (same input → same output)
- Fast computation
- Collision-resistant (hard to find two inputs with same hash)
- **Examples:** SHA-256, SHA-3, MD5 (obsolete), BLAKE2
- **Applications:**
- Password storage
- Data integrity verification
- Digital signatures
- Blockchain technology
- File checksums

D. Digital Signatures

- Provides authentication, integrity, and non-repudiation
- Like a handwritten signature but cryptographically secure
- **How it works:**

1. Hash the message
2. Encrypt hash with sender's private key (signature)
3. Recipient decrypts with sender's public key to verify

- **Examples:** RSA signatures, DSA, ECDSA
- **Applications:**
- Document signing
- Software distribution verification
- Cryptocurrency transactions

- Email authentication

E. Protocols

- Complete systems combining multiple cryptographic primitives
- **Examples:**
- **TLS/SSL:** Secure web browsing (HTTPS)
- **IPsec:** VPN security
- **SSH:** Secure remote access
- **PGP:** Email encryption
- **Kerberos:** Network authentication
- **Signal Protocol:** Encrypted messaging

2. Cryptanalysis (Breaking Codes)

Definition: The science of analyzing and breaking cryptographic systems to find weaknesses or recover encrypted information without knowing the key.

Main Categories:

A. Classical Attacks

1. Frequency Analysis

- Analyzing patterns in ciphertext based on letter/word frequency
- Effective against simple substitution ciphers
- **Example:** In English, 'E' is most common letter
- Used to break Caesar cipher, Vigenère cipher

2. Brute Force Attack

- Trying all possible keys until the correct one is found
- **Complexity:** Depends on key length
- 64-bit key: 2^64 possibilities
- 128-bit key: 2^128 possibilities
- **Defense:** Use longer keys (256-bit)

3. Known-Plaintext Attack

- Attacker has some plaintext-ciphertext pairs
- Tries to deduce the key or algorithm

4. Chosen-Plaintext Attack

- Attacker can choose plaintexts and obtain corresponding ciphertexts
- More powerful than known-plaintext attack

5. Ciphertext-Only Attack

- Attacker only has access to encrypted messages
- Most difficult type of attack

B. Modern Attacks

1. Differential Cryptanalysis

- Studies how differences in input affect differences in output
- Analyzes how the cipher propagates changes
- Effective against block ciphers
- Used to analyze DES, was considered in AES design

2. Linear Cryptanalysis

- Finds linear approximations of the cipher's behavior
- Uses linear equations to relate plaintext, ciphertext, and key bits
- Statistical attack requiring many plaintext-ciphertext pairs

3. Algebraic Attacks

- Represents cipher as system of equations
- Attempts to solve for the key
- Effective against certain stream ciphers

4. Meet-in-the-Middle Attack

- Attacks encryption schemes that use multiple rounds
- Reduces time complexity by working from both ends
- Famous for breaking 2DES (why 3DES was created)

C. Side-Channel Attacks

Definition: Exploiting physical implementation weaknesses rather than mathematical weaknesses.

Types:

1. Timing Attacks

- Measures how long operations take
- Different operations may take different times
- Can reveal information about keys or data

2. Power Analysis

- Monitors power consumption during cryptographic operations
- **Simple Power Analysis (SPA):** Direct observation
- **Differential Power Analysis (DPA):** Statistical analysis of multiple traces

- Can extract secret keys from smart cards, hardware devices

3. Electromagnetic (EM) Analysis

- Monitors electromagnetic radiation from devices
- Similar to power analysis but doesn't require physical contact

4. Acoustic Cryptanalysis

- Analyzes sounds produced by computer components
- Example: Different key presses make different sounds

5. Cache-Timing Attacks

- Exploits CPU cache behavior
- Can extract encryption keys from cloud servers

6. Fault Injection

- Deliberately causes errors in computation
- Examples: voltage glitching, laser attacks, clock manipulation
- Can bypass security or reveal keys

D. Quantum Cryptanalysis

Definition: Using quantum computers to break classical cryptographic systems.

Key Algorithms:

1. Shor's Algorithm

- Efficiently factors large numbers using quantum computers
- **Threat:** Breaks RSA, ECC, Diffie-Hellman
- Would break most current public-key cryptography
- Requires large-scale quantum computers (not yet practical)

2. Grover's Algorithm

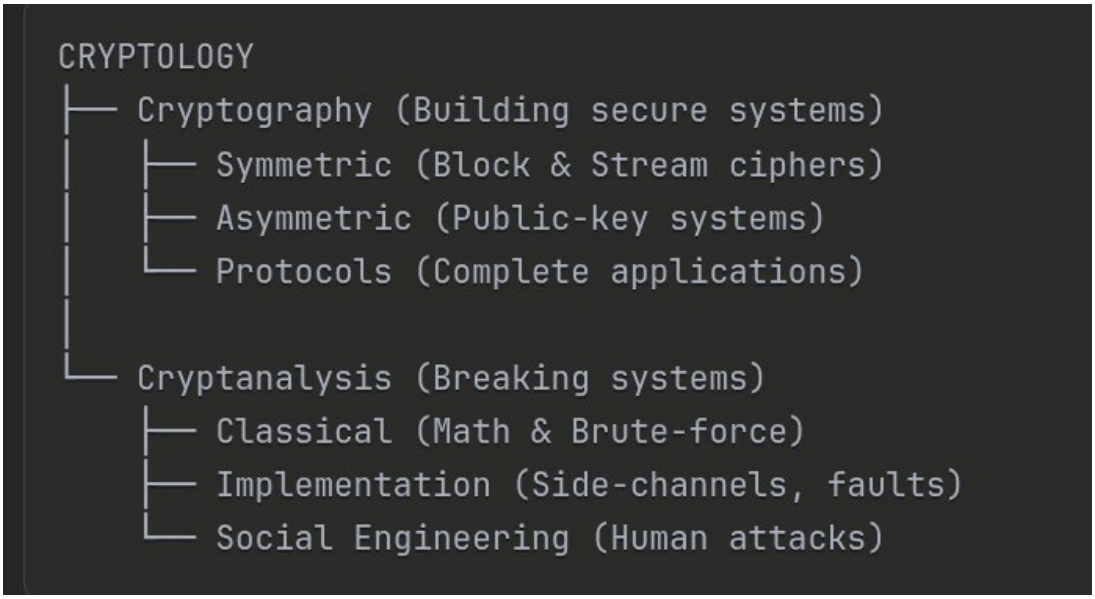
- Speeds up brute-force search
- **Impact:** Reduces symmetric key strength by half
- 256-bit key becomes effectively 128-bit
- **Defense:** Use longer keys (AES-256 instead of AES-128)

3. Post-Quantum Cryptography

- New algorithms resistant to quantum attacks
- **Examples:** Lattice-based, hash-based, code-based cryptography

- NIST is standardizing post-quantum algorithms

Image 1: Classification of the Field of Cryptology(According to slide)



This diagram shows how the field of **Cryptology** is organized:

Main Structure:

CRYPTOLOGY (top level) splits into two main branches:

1. CRYPTOGRAPHY (Left Branch)

The science of **creating** secure systems. It has three main categories:

A. Symmetric Ciphers

- Uses the **same key** for encryption and decryption
- Further divided into:
- **Block Ciphers**: Encrypts data in fixed-size blocks (e.g., 128 bits at a time)
- Examples: AES, DES, 3DES, Blowfish
- Like cutting a message into chunks and encrypting each chunk
- **Stream Ciphers**: Encrypts data bit-by-bit or byte-by-byte continuously
- Examples: RC4, ChaCha20, Salsa20
- Like encrypting data as it flows (used in real-time applications)

B. Asymmetric Ciphers

- Uses **two different keys**: public key and private key

- Examples: RSA, ECC, ElGamal
- No further subdivision shown (it's a single category)

C. Protocols

- Complete security systems that combine multiple cryptographic techniques
- Examples: TLS/SSL (HTTPS), SSH, IPsec, PGP
- These are the actual applications that use symmetric and asymmetric ciphers together

Image 2: Cryptanalysis - Attacking Cryptosystems

This diagram shows how **CRYPTANALYSIS** (the science of breaking codes) is classified:

Main Categories:

1. Classical Cryptanalysis

Attacks on the **mathematical/algorithmic** aspects of cryptography. It has two sub-types:

A. Mathematical Analysis

- Uses advanced mathematics to find weaknesses
- **Techniques include:**
- Frequency analysis (counting letter patterns)
- Differential cryptanalysis
- Linear cryptanalysis
- Algebraic attacks
- Attacks the **design** of the cipher itself

B. Brute-Force Attacks

- Simply tries **all possible keys** until finding the right one
- Example: If a cipher uses 8-bit keys, try all 256 possible combinations
- Works on any cipher but takes time
- Effectiveness depends on:
- Key length (longer = harder to break)
- Computing power available

2. Implementation Attacks

Attacks on how the cryptography is **physically implemented** (hardware/software), not the algorithm itself:

- **Side-Channel Attacks**:
- Timing attacks (measuring how long operations take)
- Power analysis (monitoring electricity usage)
- Electromagnetic analysis (reading EM radiation)

- Acoustic attacks (listening to computer sounds)
- **Fault Injection**:
- Deliberately causing errors (voltage spikes, laser attacks)
- Forcing the system to reveal secrets
- **Cache-timing attacks**:
- Exploiting CPU cache behavior

Key Point: Even if the mathematical algorithm is perfect, poor implementation can be exploited!

3. Social Engineering

Attacks on **human psychology** rather than technology:

- **Phishing**: Fake emails/websites to steal passwords
- **Pretexting**: Creating fake scenarios to trick people
- **Baiting**: Offering something attractive (free USB drive with malware)
- **Shoulder surfing**: Watching someone type their password
- **Dumpster diving**: Going through trash to find sensitive information
- **Impersonation**: Pretending to be IT support to get access

Example: Instead of breaking 256-bit encryption (nearly impossible), an attacker calls pretending to be from IT support and asks for your password (much easier!).

Famous Quote: "It's easier to trick a human than to break strong encryption."

#What is SWIFT?

SWIFT = Society for Worldwide Interbank Financial Telecommunication

What It Is

SWIFT is a **global messaging network** that allows banks and financial institutions worldwide to securely communicate and exchange financial information.

Key Points:

1. It's NOT a payment system

- SWIFT doesn't actually transfer money
- It only sends **secure messages** between banks
- Think of it as a "secure email system" for banks

2. It's a messaging protocol

- Banks send standardized messages about transactions
- Messages include: payment instructions, account details, transfer amounts
- Ensures all banks "speak the same language"

3. Global network

- Connects over **11,000+ financial institutions** in **200+ countries**
- Processes billions of messages annually
- The backbone of international banking

How It Works

Example: International Money Transfer

Let's say you want to send $1,000 from a US bank to your friend in Bangladesh:

Step-by-step:

1. **You initiate transfer** at your US bank
2. **Your bank sends SWIFT message** to recipient's bank in Bangladesh
   - Message contains: sender info, recipient info, amount, transfer instructions
1. **SWIFT network** securely transmits the message
   - Encrypted and authenticated
   - May route through intermediary banks
1. **Recipient's bank receives message**
   - Verifies the information
   - Credits your friend's account
1. **Actual money movement** happens separately
   - Banks settle accounts through correspondent banking relationships
   - This is NOT done by SWIFT itself

Analogy:

- SWIFT = Postal service (delivers the letter)
- Actual money = Package contents (separate from the message)

SWIFT Code (BIC)

What is a SWIFT Code?

Every bank has a unique **SWIFT code** (also called **BIC - Bank Identifier Code**)

Format:

AAAA BB CC DDD

- **AAAA** = Bank code (4 letters)
- **BB** = Country code (2 letters)
- **CC** = Location code (2 letters/digits)
- **DDD** = Branch code (3 letters/digits, optional)

Example:

BKBABDDH

- **BKBA** = Sonali Bank Limited
- **BD** = Bangladesh
- **DH** = Dhaka
- **(no branch code)** = Head office

When sending international money, you need the recipient's **SWIFT code** to identify their bank.

Why SWIFT is Important

1. Standardization

- All banks use the same message formats
- Reduces errors and confusion
- Enables seamless international banking

2. Security

- Encrypted messages
- Authentication of both sender and receiver
- Prevents fraud and unauthorized transactions

3. Reliability

- Highly secure and trusted network
- 99.999% uptime
- Messages delivered within seconds to minutes

4. Global Reach

- Almost every bank worldwide is connected
- Enables international trade and commerce
- Foundation of global economy

SWIFT Message Types

Common message categories:

Category	Type	Description
MT103	Customer Transfer	Standard international wire transfer
MT202	Bank Transfer	Bank-to-bank money transfer
MT700	Letter of Credit	Documentary credit issuance
MT940	Statement	Account statement message
MT950	Statement	Detailed account statement

Alternatives to SWIFT

1. CIPS (China)

- **Cross-Border Interbank Payment System**
- China's alternative to SWIFT

- Used for yuan (RMB) transactions

2. SPFS (Russia)

- **System for Transfer of Financial Messages**
- Developed after sanctions concerns
- Mainly domestic, limited international use

3. Cryptocurrency/Blockchain

- Bitcoin, Ethereum, stablecoins
- Peer-to-peer, no intermediaries
- Faster but less regulated

4. Instant Payment Systems

- **Wise (formerly TransferWise)**: Uses local banking systems
- **PayPal, Venmo**: For smaller amounts
- **RippleNet**: Blockchain-based for banks

SWIFT vs. Other Systems

Feature	SWIFT	Cryptocurrency	Instant Payment Apps
Speed	1-5 business days	Minutes to hours	Instant to 1 day
Cost	$15-50+ per transfer	$0.10-5 (network fees)	Low ($0-10)
Security	Very high (bank-level)	High (blockchain)	Medium-high
Global Reach	200+ countries	Global (internet access)	Limited countries
Regulation	Highly regulated	Varies by country	Regulated in some areas
Amount Limits	Very high (millions)	Varies	Usually lower limits

Security Aspects (Cryptography Connection)

How SWIFT Uses Cryptography:

1. Message Encryption

- All SWIFT messages are encrypted end-to-end
- Uses symmetric encryption (like AES) for speed

2. Authentication

- Digital signatures verify sender identity
- Uses asymmetric cryptography (RSA/ECC)
- Prevents impersonation

3. Message Integrity

- Hash functions ensure messages aren't tampered with
- Any modification detected immediately

4. Key Management

- Banks have unique cryptographic keys
- Keys regularly updated for security

5. Secure Network

- Multiple layers of security
- Physical security at data centers
- Network segregation

Summary

SWIFT is:

- □ A **secure messaging network** for banks
- □ The **backbone of international banking**
- □ Uses **cryptography** to ensure security
- □ **Standardizes** global financial communication

SWIFT is NOT:

- × A payment system that moves money
- × Free (banks charge fees for SWIFT transfers)
- × The only option (alternatives exist)
- × Instant (usually takes 1-5 days)

In the context of cryptography: SWIFT is a **real-world application** that heavily relies on:

- **Symmetric encryption** (for message content)
- **Asymmetric encryption** (for authentication & digital signatures)
- **Hash functions** (for integrity verification)
- **Protocols** (secure communication standards)

What is a Phishing Attack?

- A **social engineering attack** where attackers trick users into revealing sensitive info (like passwords, credit cards) by pretending to be trusted entities (banks, companies, etc.).

Types

1. **Email Phishing** – Fraudulent emails with malicious links/attachments
2. **Spear Phishing** – Targeted attacks on specific individuals/organizations
3. **Whaling** – Attacks aimed at high-level executives
4. **Smishing** – Phishing via SMS text messages
5. **Vishing** – Voice call-based phishing
6. **Clone Phishing** – Legitimate emails duplicated with malicious modifications

Key Points

- Social engineering technique
- Exploits human trust
- Common entry point for data breaches

#Why OTP is Secure

OTP = One-Time Password – A temporary code sent via SMS, email, or authenticator app for authentication.

Security Features

1. **Single-Use Only**
   - Each OTP is valid for only one login session
   - Once used, it becomes invalid immediately
   - Cannot be reused even if intercepted
1. **Time-Bound Expiration**
   - Typically expires in 30-60 seconds (TOTP - Time-based OTP)
   - Or expires after one session (HOTP - HMAC-based OTP)
   - Reduces the window of opportunity for attackers
1. **Dynamic Generation**
   - New code generated for each authentication attempt
   - Not stored anywhere permanently
   - Uses cryptographic algorithms (like HMAC)
1. **Replay Attack Prevention**
   - Even if an attacker intercepts the OTP, they cannot reuse it
   - The code becomes invalid after its first use or expiration
   - Past OTPs have no value
1. **Two-Factor Authentication (2FA)**
   - Acts as a second layer of security beyond passwords
   - Requires "something you know" (password) + "something you have" (OTP device)
   - Significantly reduces risk of unauthorized access
1. **Phishing Resistance**
   - Even if password is compromised, attacker still needs the OTP
   - OTP is delivered through a separate channel (phone/app)

How It Works

- Generated using algorithms like TOTP (Time-based) or HOTP (Counter-based)
- Synchronized between server and client
- Verified once and discarded

What is a Protocol?

Definition: A set of rules and standards that govern how data is transmitted, formatted, and processed between devices in a network.

Purpose

- Ensures devices can communicate regardless of manufacturer
- Defines data format, transmission timing, error handling, and authentication
- Establishes common "language" for network communication

Types of Protocols by Layer

1. Application Layer

- **HTTP/HTTPS** – Web browsing (HTTPS is secure)
- **FTP/SFTP** – File transfer
- **SMTP** – Email sending
- **DNS** – Domain name resolution
- **SSH** – Secure remote access

2. Transport Layer

- **TCP** – Reliable, connection-oriented (guarantees delivery)
- **UDP** – Fast, connectionless (no delivery guarantee)

3. Network Layer

- **IP** – Addressing and routing (IPv4, IPv6)
- **ICMP** – Error reporting and diagnostics (ping)

4. Data Link Layer

- **Ethernet** – Wired LAN communication
- **PPP** – Point-to-point connections

5. Security Protocols

- **SSL/TLS** – Encrypts data in transit (web security)
- **IPsec** – Secures IP communications (VPNs)
- **Kerberos** – Network authentication
- **Diffie-Hellman** – Secure key exchange
- **SSH** – Encrypted remote login

Key Characteristics

- **Standardized** – Defined by organizations (IETF, IEEE, W3C)
- **Interoperable** – Works across different systems
- **Layered** – Each protocol operates at specific network layer

HTTP vs HTTPS

HTTP (Hypertext Transfer Protocol)

- **Unsecured protocol** for web communication
- Data transmitted in **plain text** (readable by anyone)
- Uses **Port 80**
- No encryption or authentication
- Vulnerable to eavesdropping and data theft

HTTPS (HTTP Secure)

- **Secured version** of HTTP
- Uses **SSL/TLS encryption** to protect data
- Uses **Port 443**
- Requires **digital certificate** (issued by Certificate Authority)
- Standard for banking, e-commerce, login pages

Why HTTPS is More Secure

1. Encryption

- Data encrypted during transmission
- Unreadable to interceptors
- Protects passwords, credit cards, personal info

2. Authentication

- Verifies server identity through SSL/TLS certificate
- Confirms you're connected to the legitimate website
- Prevents fake/spoofed websites

3. Data Integrity

- Detects if data is tampered during transit
- Ensures data arrives unchanged
- Uses cryptographic hashing

4. Attack Prevention

- **Man-in-the-Middle (MITM)** – Attacker cannot read encrypted data
- **Eavesdropping** – Traffic is protected from sniffing
- **Session Hijacking** – Harder to steal session cookies

Key Differences Summary

Feature	HTTP	HTTPS
Security	None	SSL/TLS encrypted
Port	80	443
Data Format	Plain text	Encrypted
Certificate	Not required	Required
Use Case	Non-sensitive content	Sensitive transactions

Visual Indicator

- **HTTPS** shows a **padlock icon** 🔒 in browser address bar
- Builds user trust and confidence

#Blockchain

Definition: A decentralized, distributed digital ledger that securely records transactions across multiple computers in a network.

Key Features

1. Decentralization

- No central authority or single point of control
- Data stored across multiple nodes (computers)
- Eliminates need for intermediaries (banks, brokers)

2. Transparency

- All transactions are visible to network participants
- Every node has a copy of the entire ledger
- Increases accountability and trust

3. Immutability

- Once data is recorded, it cannot be altered or deleted
- Each block is cryptographically linked to the previous one
- Tampering with one block invalidates the entire chain

4. Security

- Uses **cryptographic hashing** (SHA-256) to secure data
- **Digital signatures** verify transaction authenticity
- **Public-key cryptography** for user identification

5. Consensus Mechanisms

- Network agrees on validity of transactions
- Common methods: Proof of Work (PoW), Proof of Stake (PoS)
- Prevents double-spending and fraud

How It Works

1. **Transaction Initiated** – User requests a transaction
2. **Broadcast to Network** – Transaction sent to all nodes
3. **Validation** – Nodes verify transaction using consensus
4. **Block Creation** – Verified transactions grouped into a block
5. **Hashing** – Block gets unique cryptographic hash
6. **Chain Addition** – New block linked to previous block
7. **Distribution** – Updated blockchain distributed to all nodes

Core Components

- **Block**: Contains transaction data, timestamp, hash, and previous block's hash
- **Chain**: Linked sequence of blocks
- **Nodes**: Computers maintaining copies of the blockchain
- **Hash**: Unique fingerprint of each block (ensures integrity)

Applications

- Cryptocurrencies (Bitcoin, Ethereum)
- Supply chain tracking
- Digital identity verification
- Smart contracts
- Medical records management
- Voting systems

Security Benefits

- **Data Integrity**: Hashing ensures data hasn't been tampered with
- **Distributed Storage**: No single point of failure
- **Transparency**: All changes are traceable
- **Authentication**: Digital signatures verify participants

#Red Telephone → Dedicated Line

- A "**dedicated secure line**" used for communication between two trusted parties (like the famous red telephone between the U.S. and Russia).
- In cryptography context: it symbolizes a **secure communication channel**, similar to a **key exchange** over a protected medium.

#Encryption & Decryption Equations

- x = plaintext
- y = ciphertext
- K = key
- Both use the same key in symmetric cryptography.

• Encryption equation	$y = e_K(x)$
• Decryption equation	$x = d_K(y)$

- Encryption and decryption are inverse operations if the same key K is used on both sides:

$$d_K(y) = d_K(e_K(x)) = x$$

WiFi Security

Definition: Protection of wireless networks from unauthorized access, eavesdropping, and cyberattacks using encryption protocols and authentication mechanisms.

WiFi Security Protocols

1. WEP (Wired Equivalent Privacy)

- **Status**: Obsolete and insecure
- Uses RC4 encryption algorithm
- Easily cracked within minutes
- **Do not use** – highly vulnerable

2. WPA (Wi-Fi Protected Access)

- Developed to replace WEP
- Uses **TKIP (Temporal Key Integrity Protocol)**
- Better than WEP but still has vulnerabilities
- Transitional protocol

3. WPA2 (Wi-Fi Protected Access 2)

- **Current standard** (since 2004)
- Uses **AES (Advanced Encryption Standard)** encryption
- Strong symmetric encryption (128-bit or 256-bit)
- Two modes:
- **WPA2-Personal (PSK)** – Pre-Shared Key for home/small networks
- **WPA2-Enterprise** – Uses RADIUS server for authentication
- Vulnerable to **KRACK attack** (Key Reinstallation Attack)

4. WPA3 (Wi-Fi Protected Access 3)

- **Latest and most secure** (2018)
- Protects against brute-force password attacks
- Uses **SAE (Simultaneous Authentication of Equals)** instead of PSK
- Provides **forward secrecy** – past data safe even if password compromised
- Better protection on public/open networks

Why Do We Need Cryptanalysis?(CT)

Definition: The study and practice of analyzing cryptographic systems to find vulnerabilities and test their security strength.

Primary Reasons

1. No Mathematical Proof of Security

- No practical cipher has a mathematical guarantee of being unbreakable
- Theoretical security ≠ practical security
- Cannot prove a cipher is secure through math alone

2. Security Through Testing

- The only way to gain confidence in a cipher's security is to **attempt to break it and fail**
- Continuous testing by skilled cryptanalysts strengthens assurance
- If experts cannot break it after years of trying, it's likely secure

3. Kerckhoff's Principle

- A cryptosystem should be secure even if the attacker knows all details about the system, **except the secret key**
- Security must not rely on secrecy of the algorithm
- Only the key should remain secret

4. Validating Cipher Design

- Tests real-world resistance against various attack methods
- Identifies weaknesses before deployment
- Ensures cipher works as intended under adversarial conditions

5. Preventing "Security by Obscurity"

- Secret ciphers historically fail once reverse-engineered
- Example: **Content Scrambling System (CSS)** for DVD protection was broken after being reverse-engineered
- Public scrutiny makes ciphers stronger, not weaker

Best Practice

Only use widely known ciphers that have been:

- Publicly available for cryptanalysts to examine
- Tested for several years by expert cryptographers
- Survived multiple attack attempts
- Standardized by recognized organizations (NIST, ISO)

Types of Attacks Tested

1. **Mathematical Analysis** – Finding algorithmic weaknesses
2. **Brute-Force Attack** – Exhaustive key search
3. **Implementation Attacks** – Exploiting side channels (power consumption, timing)
4. **Social Engineering** – Tricking users into revealing keys

Key Takeaway

Cryptanalysis is **essential quality assurance** for cryptography. A cipher is only as secure as the attacks it has successfully withstood.

> **Simply??**
>
> We need cryptanalysis because there is no guaranteed mathematical proof that any cipher is completely secure.
> Through cryptanalysis, experts test and attempt to break encryption systems to find weaknesses or vulnerabilities.
> If a cipher resists all known attacks, it is considered strong and reliable for practical use.

Kerckhoff's Principle states that a cryptosystem should remain secure even if everything about it is public, except the secret key.

In practice, this means:

- We should use only well-known and tested ciphers that have been analyzed by experts for many years.
- Keeping the algorithm secret does not make it more secure — real security depends only on keeping the key secret.

Example: The CSS (Content Scrambling System) used for DVD protection was broken easily because it relied on secrecy instead of strong cryptographic design.

When Is a Cipher More Secure?

A cipher is considered **more secure** when it meets the following points:

1. **Large Key Size:**
   – A bigger key makes brute-force attacks practically impossible, as it increases the number of possible key combinations.
2. **Strong Encryption Algorithm:**
   – It should use a well-designed and mathematically strong algorithm that has been publicly tested and analyzed by experts.
3. **Random Ciphertext Output:**
   – The ciphertext should appear completely random with no visible patterns that could help attackers guess the plaintext.
4. **Resistance to Known Attacks:**
   – The cipher must resist all common attacks such as brute-force, frequency analysis, mathematical, and side-channel attacks.
5. **No Known Weaknesses:**
   – There should be no publicly known vulnerabilities or design flaws in the algorithm.
6. **Proper Implementation:**
   – Even a strong cipher can fail if implemented poorly. Secure software and hardware implementation are essential.
7. **Regular Testing and Review:**
   – The cipher should be continuously analyzed and tested by cryptographers to ensure it remains secure over time.

#Reverse Engineering & CSS (Content Scrambling System)

Reverse Engineering: The process of analyzing a product, system, or algorithm to understand how it works without access to its original design documentation.

Purpose of Reverse Engineering

- Discover how proprietary systems function
- Find security vulnerabilities
- Replicate or improve designs
- Break encryption schemes

CSS (Content Scrambling System) Case Study

What is CSS?

- **Encryption system** designed to protect DVD video content from piracy
- Developed in the 1990s for DVD copy protection
- Algorithm details kept **secret** (security through obscurity)

How CSS Failed

1. **Secret Algorithm Approach**
   - CSS designers kept the encryption method hidden
   - Believed secrecy would provide security
   - **Violated Kerckhoff's Principle**
1. **Reverse Engineering Success**
   - Hackers analyzed DVD players to understand CSS
   - Extracted and decoded the algorithm
   - Published the method publicly (DeCSS tool in 1999)
1. **Complete Break**
   - Once reversed, CSS was found to have **weak encryption** (40-bit key)
   - Could be cracked in seconds
   - DVD content protection became ineffective

Key Lessons from CSS Failure

1. **Secrecy ≠ Security**
   - Hiding the algorithm doesn't make it secure
   - Once discovered, weak design is exposed
1. **Validation of Kerckhoff's Principle**
   - Security should rely on key secrecy, not algorithm secrecy

- Public algorithms undergo proper scrutiny

1. **Weak Design Cannot Be Hidden**
   - CSS used inadequate key length (40-bit)
   - Poor cryptographic design was masked by secrecy
   - Would have been caught early if publicly reviewed

Modern Approach

Properly designed systems:

- Use **public, well-tested algorithms** (AES, RSA)
- Rely on **strong keys** (128-bit, 256-bit)
- Allow independent security audits
- Follow **Kerckhoff's Principle**

Bottom Line

The CSS example proves that **keeping algorithms secret eventually fails**. True security comes from strong, publicly vetted cryptographic design with secret keys.

#Modular Arithmetic – Complete Detailed Notes

1. Introduction to Modular Arithmetic

Modular arithmetic is a mathematical system where numbers "wrap around" after reaching a certain value known as the *modulus*.
It is also called **clock arithmetic**, because numbers repeat cyclically, like hours on a clock.

Example (Clock Analogy):
A clock has 12 hours. After 12 comes 1 again.
So:
$14 \equiv 2 \pmod{12}$
We stay within a finite set of numbers — from 1 to 12.

2. Why Do We Need to Study Modular Arithmetic?

1. **Fundamental for Modern Cryptography**
   - It is the foundation for asymmetric cryptography systems like:
     • RSA

- Diffie–Hellman Key Exchange
- Elliptic Curve Cryptography (ECC)

- These rely on modular exponentiation and modular inverses.

1. **Used in Classical Ciphers**

- Historical ciphers such as Caesar Cipher and Affine Cipher are elegantly expressed using modular arithmetic.

1. **Basis of Modern Cryptosystems**

- Cryptosystems operate on sets of numbers that are:
  - *Discrete* – integer-based, not continuous
  - *Finite* – operations occur within limited sets of numbers

1. **Keeps Numbers Within Limits**

- Ensures all encryption/decryption results remain inside a fixed range (like bytes in digital systems).

3. Definition: Modulus Operation

Let a, r, and m be integers, with $m > 0$.
We write:

$a \equiv r \pmod{m}$

if and only if $(r - a)$ is divisible by m.

That means:
m divides $(r - a)$

Terms:
- m = modulus
- r = remainder (or residue)

4. Examples of Modular Reduction

1. $a = 12, m = 9 \rightarrow 12 \equiv 3 \pmod{9}$
   (Because $12 - 3 = 9$ is divisible by 9)
2. $a = 37, m = 9 \rightarrow 37 \equiv 1 \pmod{9}$
   (Because $37 - 1 = 36$ is divisible by 9)
3. $a = -7, m = 9 \rightarrow -7 \equiv 2 \pmod{9}$
   (Because $-7 - 2 = -9$ is divisible by 9)

In all cases, m divides $(a - r)$.

5. The Remainder Is Not Unique

For any modulus m, a number a can have infinitely many valid remainders because you can add or subtract multiples of m.

Example:
For a = 12, m = 9:
• $12 \equiv 3 \pmod 9$
• $12 \equiv 21 \pmod 9$
• $12 \equiv -6 \pmod 9$

All are valid because:
• $12 - 3 = 9$
• $12 - 21 = -9$
• $12 - (-6) = 18$
All are divisible by 9.

6. Which Remainder Do We Choose?

By convention, we use the **smallest non-negative remainder**.

Formula:
$a = q \times m + r$, where $0 \leq r < m$
(q = quotient, r = remainder)

Example:
a = 12, m = 9
$12 = 1 \times 9 + 3 \rightarrow r = 3$
Hence, the *canonical remainder* is 3.

7. Modular Division and Multiplicative Inverse

Problem:
Direct division is not defined in modular arithmetic.

Solution:
We multiply by the **modular inverse** instead.

Formula:
$b / a \equiv b \times a^{-1} \pmod m$

where a^{-1} is the modular inverse of a, defined as:
$a \times a^{-1} \equiv 1 \pmod m$

Example: Find (5 / 7) mod 9

Step 1: Find the inverse of 7 mod 9.
We need $7 \times x \equiv 1 \pmod 9$.

Try x = 1, 2, 3, 4 …

x	7×x	7×x mod 9
1	7	7
2	14	5
3	21	3
4	28	1 □

→ So $7^{-1} \equiv 4 \pmod 9$

Step 2: Multiply by 5.
$5 / 7 \equiv 5 \times 4 = 20 \equiv 2 \pmod 9$

□ **Answer:** $5 / 7 \equiv 2 \pmod 9$

When Does an Inverse Exist?

An inverse exists **only if gcd(a, m) = 1**,
that is, when a and m are *coprime*.

a	m	gcd(a, m)	Inverse Exists?
7	9	1	□ Yes
5	9	1	□ Yes
3	9	3	× No
6	9	3	× No

How to Find the Inverse

1. Exhaustive Search (Trial Method)
Try x = 1 to m − 1 until $a \times x \equiv 1 \pmod m$.

2. Extended Euclidean Algorithm (Efficient Method)
For large numbers (like in RSA), use this algorithm to find x, y such that:
$a \times x + m \times y = 1$
Then, x is the modular inverse of a mod m.

8. Modular Reduction During Calculations

You can reduce intermediate results at any step to simplify computations.

Example: Compute $3^8 \bmod 7$

Method 1 (Direct):
$3^8 = 6561 \rightarrow 6561 \bmod 7 = 2$

Method 2 (Reduce Intermediately):
$3^2 = 9 \equiv 2 \pmod 7$

$3^4 = (3^2)^2 = 2^2 = 4 \pmod 7$
$3^8 = (3^4)^2 = 4^2 = 16 \equiv 2 \pmod 7$

 Same result (2), easier computation.

Rule:
Reduce intermediate results early to keep numbers small.

9. Algebraic View – The Ring Zm

Modular arithmetic operates in a set called the **integer ring Zm**, which contains all integers from 0 to m − 1.

Zm = {0, 1, 2, 3, ..., m − 1}

Properties of Zm:

1. **Closure:**
   a + b ∈ Zm, and a × b ∈ Zm
2. **Associativity:**
   a + (b + c) = (a + b) + c,
   a × (b × c) = (a × b) × c
3. **Commutativity:**
   a + b = b + a,
   a × b = b × a
4. **Distributive Law:**
   a × (b + c) = (a × b) + (a × c)
5. **Additive Identity:**
   a + 0 ≡ a (mod m)
6. **Additive Inverse:**
   For every a, there exists −a such that a + (−a) ≡ 0 (mod m)
7. **Multiplicative Identity:**
   a × 1 ≡ a (mod m)
8. **Multiplicative Inverse:**
   Exists only if gcd(a, m) = 1

Example: The Ring Z9 = {0,1,2,3,4,5,6,7,8}

Element (a)	Inverse (a^{-1} mod 9)	Exists?
0	None	×
1	1	
2	5	
3	None	×

4	7	□
5	2	□
6	None	×
7	4	□
8	8	□

Only numbers coprime with 9 have inverses.

10. Key Takeaways

Concept	Explanation
Definition	$a \equiv r \pmod m$ means m divides $(a - r)$
Modulus (m)	The "wrap-around" base
Remainder (r)	Result after division
Inverse Exists When	$\gcd(a, m) = 1$
Division Rule	$b / a \equiv b \times a^{-1} \pmod m$
Canonical Remainder	Smallest positive remainder $(0 \leq r < m)$
Algebraic Structure	Ring Zm
Best Practice	Reduce intermediate results early
Applications	RSA, ECC, Caesar, Affine Cipher

11. Importance in Cryptography

1. **RSA Algorithm**
   - Encryption: $C = M^e \bmod n$
   - Decryption: $M = C^d \bmod n$
1. **Elliptic Curve Cryptography (ECC)**
   - Point addition and multiplication are done modulo a prime number.
1. **Classical Ciphers**
   - Caesar cipher → shift letters mod 26
   - Affine cipher → uses modular inverse of key a mod 26
1. **Efficiency**
   - Modular reduction keeps numbers manageable.

□ Summary

- Modular arithmetic = “wrap-around” arithmetic
- $a \equiv r \pmod m$ → both have the same remainder when divided by m
- Multiple remainders are valid (differ by multiples of m)
- Canonical remainder = smallest non-negative remainder
- Division → multiply by modular inverse
- Inverse exists only if $\gcd(a, m) = 1$
- Zm is a ring that supports +, −, ×
- Core mathematical concept for **modern cryptography**

1. Shift (or Caesar) Cipher

□ What It Is

The **Shift Cipher** (also called **Caesar Cipher**) is one of the oldest and simplest encryption techniques in history.
It was allegedly used by **Julius Caesar** to send secret messages to his generals.

It works by **shifting the letters** of the alphabet by a fixed number (**k**) positions.

□ Step 1: Mapping Letters to Numbers

We first assign a number to each letter:

Letter	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
Number	0	1	2	3	4	5	6	7	8	9	10	11	12
Letter	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Number	13	14	15	16	17	18	19	20	21	22	23	24	25

So, A = 0, B = 1, C = 2, ..., Z = 25.

□ Step 2: Encryption Formula

The encryption shifts each letter by *k* positions:

y = (x + k) mod 26

- x → number of plaintext letter
- y → number of ciphertext letter
- mod 26 → because there are 26 letters in the alphabet

□ Example:

Let's encrypt **ATTACK** with key **k = 7**

Plaintext letters → convert to numbers:
A T T A C K
0 19 19 0 2 10

Now apply the formula y = (x + 7) mod 26

Plaintext (x)	**+7**	**Cipher (y)**	**Cipher Letter**
0	7	7	H

19	26 0	A
19	26 0	A
0	7 7	H
2	9 9	J
10	17 17	R

Ciphertext = **HAAHJR**

□ Wrapping Around (mod 26)

Notice that if adding k goes beyond 25 (Z), it "wraps around" back to A.

Example:

19 + 7 = 26 ≡ 0 mod 26 → T becomes A

This *wrap-around* behavior is what modular arithmetic handles perfectly.

□ Decryption Formula

To decrypt, just subtract the shift:

x = (y − k) mod 26

So if you know k = 7, you simply shift each ciphertext letter 7 positions *backward*.

□ Security of the Shift Cipher

Is the Shift Cipher secure?
→ × No, it's *not secure at all.*

Reasons:

1. **Tiny key space:** Only 26 possible keys (k = 0–25).
   → A computer (or even a human) can try all possibilities in seconds.
2. **Letter frequency analysis:**
   Each letter in the ciphertext corresponds to one plaintext letter.
   So attackers can guess the mapping by comparing how often each letter appears (E is common in English, etc.).

□ 2. Affine Cipher

The **Affine Cipher** is an improvement on the Shift Cipher — it adds a *multiplication* step before shifting.

□ Formula

Encryption:

$y = (a \times x + b) \bmod 26$

Decryption:

$x = a^{-1} \times (y - b) \bmod 26$

- **a** and **b** together form the **key**: k = (a, b)
- **a^{-1}** is the *modular inverse* of a (so that $a \times a^{-1} \equiv 1 \bmod 26$)
- x = plaintext letter number
- y = ciphertext letter number

How It Works

The affine cipher first multiplies the letter's number (x) by *a*, then shifts it by *b*, all modulo 26.

It's like combining:

- A **multiplicative cipher** (multiply by a)
- A **shift cipher** (add b)

Example

Let's encrypt with key (a, b) = (5, 8).

Formula: $y = (5x + 8) \bmod 26$

Plaintext: "C" → x = 2

$y = (5 \times 2 + 8) \bmod 26 = (10 + 8) \bmod 26 = 18 \bmod 26 = 18$
→ Ciphertext letter = S

So "C" encrypts to "S".

Decryption

To decrypt, we use:

$x = a^{-1} \times (y - b) \bmod 26$

We need the **inverse of a** under mod 26.

For a = 5,
find a^{-1} such that $5 \times a^{-1} \equiv 1 \bmod 26$
→ $a^{-1} = 21$ (since $5 \times 21 = 105 \equiv 1 \bmod 26$)

Then, plug values into formula to recover x.

□ Valid Values for "a"

We can only use values of **a** that have an inverse mod 26.
This means **gcd(a, 26) = 1** (they must be coprime).

The valid values of a are:
{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25} → **12 possible a values**

Since b can be any number from 0–25,
Total key combinations = 12 × 26 = 312 possible keys.

□ Security

The affine cipher is **slightly better** than the shift cipher — but still **weak**.

Attacks possible:

1. **Exhaustive key search:** Only 312 keys — very small space.
2. **Frequency analysis:** Still a simple substitution cipher (one plaintext letter maps to one ciphertext letter).

So, modern cryptography **never uses it for real security** — it's only useful for learning concepts.

□ 3. Lessons Learned (Very Important)

These points summarize the philosophy of **modern cryptography**:

□ 1. Never invent your own cipher

Unless you are a trained cryptographer, **do not design your own algorithm.**
Even simple-looking ciphers can have hidden weaknesses.

Example:
The DVD encryption system **CSS** was secret but was easily broken after reverse engineering.

□ 2. Always use proven algorithms

Only use algorithms that have been:

- Publicly analyzed
- Tested for years by professional cryptanalysts

Examples:

- AES (Advanced Encryption Standard)
- RSA
- ECC

□ 3. Attackers exploit the weakest link

Even if a cipher has a large key (like 256-bit), if there's an analytical or side-channel weakness, it can be broken.

Security = strength of the **whole system**, not just key length.

□ 4. Key lengths and security levels

Approximate guidelines:

Key Length (bits)	Security Level	Notes
64 bits	× Insecure	Too small for modern data
128 bits	□ Secure for decades	Good for long-term security
256 bits	🔒 Stronger, even against quantum attacks	Future-proof

(Quantum computers don't exist yet — but 256-bit keys are considered safe if they ever do.)

□ 5. Role of Modular Arithmetic

- Modular arithmetic provides a **mathematical foundation** for encryption.
- It allows operations like:
- "wrap-around" addition
- multiplicative inverses
- modular exponentiation (used in RSA, ECC)

Example:
Affine cipher → $y = a \times x + b \pmod{26}$
RSA → $C = M^e \pmod{n}$

So, modular arithmetic makes encryption formulas **work within fixed limits**, keeping numbers bounded and reversible.

□ Summary Table

Concept	Shift Cipher	Affine Cipher
Formula	$y = (x + k) \bmod 26$	$y = (a \times x + b) \bmod 26$
Key	Single value (k)	Pair (a, b)
Valid Keys	26 possible	312 possible
Inverse Needed?	No	Yes ($a^{-1} \bmod 26$)
Security	Very weak	Slightly better, still weak
Attack Type	Brute-force, frequency	Same (plus easier math attacks)

□ Final Intuition

- The **Shift Cipher** simply shifts letters by a fixed amount.
- The **Affine Cipher** multiplies and then shifts.
- Both use **mod 26 arithmetic** to keep letters in the alphabet range (A–Z).
- Both are **classical**, easy to understand, but **not secure**.
- Modern cryptography builds upon these ideas using **large numbers, modular exponentiation, and mathematical hardness**.