ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

INTRODUCTION TO CRYPTOGRAPHY (AES, RSA, HASHING ALGORITHMS)

📌 CHAPTER 1: INTRODUCTION TO CRYPTOGRAPHY

1.1 What is Cryptography?

Cryptography is the **study and practice of techniques** used to protect the **confidentiality, integrity, and authenticity of information**. It involves encoding messages so that only authorized parties can access and interpret them.

In the modern digital era, cryptography is used in **banking transactions, secure communications, password protection, blockchain technology, and many more applications**.

1.2 The Evolution of Cryptography

Cryptography has evolved from simple **manual ciphers** to complex **mathematical algorithms** used in computing today.

📌 Historical Cryptographic Methods:

Time Period	Method Used	Description

Ancient Times	**Caesar Cipher**	A simple shift cipher used by Julius Caesar.
9th Century	**Arabic Cryptanalysis**	Al-Kindi developed frequency analysis for breaking ciphers.
16th Century	**Vigenère Cipher**	A polyalphabetic substitution cipher.
20th Century	**Enigma Machine**	Used in World War II, broken by Alan Turing's team.
Modern Era	**AES, RSA, ECC**	Advanced encryption used in cybersecurity and data protection.

1.3 Objectives of Cryptography

Cryptography serves four main purposes, known as the **CIA Triad**:

Objective	Description
Confidentiality	Ensures only authorized users can access data.
Integrity	Ensures data is not altered during transmission.
Authentication	Verifies the sender's identity.
Non-Repudiation	Prevents a sender from denying they sent a message.

📌 Example: How Cryptography Works in Online Banking

- When logging into an **online banking system, AES encryption** protects your password.
- **RSA encryption** is used to establish a secure connection between your browser and the bank's server.

- **Hashing** ensures that transaction details are **not tampered with**.

📌 Chapter 2: Types of Cryptography

Cryptography is broadly classified into **three categories**:

🖼️ Diagram: Types of Cryptography

Cryptography

├─── Symmetric Encryption (AES, DES)

├─── Asymmetric Encryption (RSA, ECC)

└─── Hashing (SHA-256, MD5)

2.1 Symmetric Encryption

✓ Uses a **single secret key** for both encryption and decryption.
✓ Faster and more efficient than asymmetric encryption.
✓ The biggest challenge is **securely sharing the secret key**.

📌 Examples of Symmetric Encryption:

- **AES (Advanced Encryption Standard)** – Used for encrypting files, Wi-Fi security.
- **DES (Data Encryption Standard)** – An older encryption method, now considered insecure.

2.2 Asymmetric Encryption (Public Key Cryptography)

✓ Uses **two keys** – a **public key (for encryption)** and a **private key (for decryption)**.

✓ More secure than symmetric encryption but **slower due to complex calculations**.

📌 **Examples of Asymmetric Encryption:**

- **RSA (Rivest-Shamir-Adleman)** – Used in SSL certificates, secure email.
- **ECC (Elliptic Curve Cryptography)** – More efficient than RSA, used in mobile devices.

2.3 Hashing

✓ **One-way function** – Converts data into a fixed-length hash value.
✓ Hashes **cannot be reversed** (unlike encryption).

📌 **Examples of Hashing Algorithms:**

- **SHA-256 (Secure Hash Algorithm)** – Used in Bitcoin and blockchain security.
- **MD5 (Message Digest Algorithm)** – Previously used for password hashing but now considered weak.

📌 Chapter 3: Symmetric Encryption – AES

3.1 What is AES?

AES is a **modern encryption standard** that was adopted in **2001** by the **U.S. government** to replace **DES** due to its vulnerabilities. It is widely used for **securing sensitive data**.

📌 **Features of AES:**

✓ Uses **128-bit, 192-bit, or 256-bit** key sizes.

✓ Encrypts data **in fixed 128-bit blocks**.
✓ Resistant to **brute-force attacks**.

3.2 How AES Works

AES performs **multiple rounds of encryption,** depending on the key size:

- **AES-128 → 10 rounds**
- **AES-192 → 12 rounds**
- **AES-256 → 14 rounds**

📌 **AES Steps:**

1. **SubBytes** – Replaces bytes using a substitution table.
2. **ShiftRows** – Shifts rows of the data matrix.
3. **MixColumns** – Scrambles data to increase security.
4. **AddRoundKey** – XORs the data with a secret key.

📌 CHAPTER 4: ASYMMETRIC ENCRYPTION – RSA

4.1 What is RSA?

The **Rivest-Shamir-Adleman (RSA)** algorithm is an asymmetric encryption method **invented in 1977**.

📌 **RSA Features:**
✓ Uses **two keys** (public and private).
✓ Key sizes **start at 1024-bit** (recommended: **2048-bit or 4096-bit**).
✓ Used in **digital signatures, SSL certificates, secure messaging**.

4.2 How RSA Works

📌 **Steps of RSA Encryption & Decryption:**

1. **Key Generation** – Public-private key pair is generated.
2. **Encryption** – Sender encrypts a message with the recipient's public key.
3. **Decryption** – Recipient decrypts the message with their private key.

🖼️ **Diagram: RSA Encryption & Decryption**

Plaintext ⟶ Public Key (Encryption) ⟶ Ciphertext ⟶ Private Key (Decryption) ⟶ Plaintext

✓ **Use Cases:**

- **Secure websites (HTTPS/TLS)**
- **Email encryption (PGP, S/MIME)**
- **Blockchain and cryptocurrency transactions**

📌 Chapter 5: Hashing Algorithms

5.1 What is Hashing?

Hashing is a **one-way function** that converts data into a **fixed-length hash value**. It is used for:

✓ **Password security**

✓ **Data integrity checks**

✓ **Blockchain technology**

5.2 Popular Hashing Algorithms

Algorithm	Hash Length	Use Case
MD5	128-bit	Obsolete (Weak security)
SHA-256	256-bit	Blockchain, digital signatures
SHA-3	512-bit	Secure password hashing
Bcrypt	Variable	Secure password storage

📌 Chapter 6: Summary & Next Steps

✅ Key Takeaways

- **AES (Symmetric)** – Fast encryption for data security (Wi-Fi, file encryption).
- **RSA (Asymmetric)** – Uses public-private keys (Secure emails, HTTPS).
- **Hashing (SHA-256, Bcrypt)** – Converts data into a fixed-length irreversible format (Passwords, Blockchain).

🚀 NEXT STEPS

- Learn **cryptographic attacks** (brute force, side-channel attacks).
- Experiment with **hybrid encryption (AES + RSA for secure messaging)**.
- Understand **how cryptography is used in blockchain and cryptocurrencies**.

PUBLIC KEY INFRASTRUCTURE (PKI) & DIGITAL CERTIFICATES

📌 CHAPTER 1: INTRODUCTION TO PUBLIC KEY INFRASTRUCTURE (PKI)

1.1 What is Public Key Infrastructure (PKI)?

Public Key Infrastructure (PKI) is a **security framework** that enables **secure communication, authentication, and encryption** using cryptographic key pairs (public and private keys) and digital certificates. It is a **trust model** that organizations use to **establish the identities of users, devices, and services** over a network.

PKI plays a crucial role in securing **web browsing (HTTPS), email encryption (S/MIME), digital signatures, VPNs, and authentication mechanisms** in modern IT environments.

📌 Core Functions of PKI:

✓ **Confidentiality** – Encrypts communication, ensuring only authorized parties can read the data.
✓ **Integrity** – Prevents unauthorized modification of data through hashing and digital signatures.
✓ **Authentication** – Ensures the identity of users, systems, or organizations.
✓ **Non-Repudiation** – Ensures that senders cannot deny sending a message or signing a document.

1.2 Importance of PKI in Cybersecurity

Without PKI, sensitive communications could be intercepted, altered, or forged. PKI provides **a structured way to issue, validate, and revoke digital certificates**, ensuring the **trustworthiness** of communications.

🚨 **Real-World Example:**

- **Online Banking:** PKI ensures that login credentials and transactions remain encrypted and authenticated.
- **Government Services:** Digital signatures verify the authenticity of official documents (e.g., e-passports, Aadhaar authentication).
- **E-Commerce Transactions:** SSL/TLS certificates ensure secure payments in online shopping.

🖼 **Diagram: How PKI Works**

User Requests a Digital Certificate → Certificate Authority (CA) Verifies Identity → Issues Certificate → Used for Secure Communication & Authentication

---

📌 Chapter 2: Components of Public Key Infrastructure (PKI)

PKI consists of several critical components that work together to create a **secure communication system**.

2.1 Certificate Authority (CA)

A **Certificate Authority (CA)** is a **trusted entity** that issues and manages digital certificates. The CA acts as the **root of trust**, ensuring that individuals, organizations, and websites are authentic before issuing them a certificate.

📌 **Types of CAs:**

✓ **Root CA:** The highest authority in the PKI hierarchy.

✓ **Intermediate CA:** Acts as a bridge between the Root CA and end-users.

✓ **Public CA:** Trusted by external users (e.g., DigiCert, Let's Encrypt).

✓ **Private CA:** Used within organizations for internal security.

📌 **Functions of a CA:**

- Verifies the identity of users, websites, and organizations.
- Issues digital certificates containing the public key.
- Maintains certificate validity and revokes compromised certificates.

2.2 Registration Authority (RA)

A **Registration Authority (RA)** acts as a **verifying entity** that **checks the credentials** of a certificate applicant before a CA issues a certificate.

📌 **Functions of RA:**

✓ Verifies identities before the CA grants certificates.

✓ Approves or denies certificate requests.

✓ Ensures compliance with security policies.

2.3 Public & Private Keys

PKI relies on **asymmetric encryption**, using a pair of cryptographic keys:

Key Type	Purpose
Public Key	Available to anyone, used for **encryption and signature verification**
Private Key	Kept secret, used for **decryption and signing**

📌 **How Key Pairs Work:**

1. Alice encrypts a message using Bob's **public key**.
2. Bob decrypts the message using his **private key**.

2.4 Digital Certificates

A **digital certificate** is an electronic document issued by a CA that binds a **public key to a verified entity** (e.g., a person, company, or website).

📌 **Certificate Contents:**

✓ Public key of the owner.
✓ Name of the certificate holder.
✓ Expiration date of the certificate.
✓ Digital signature of the CA.

2.5 Certificate Revocation List (CRL) & Online Certificate Status Protocol (OCSP)

- **CRL:** A list of **revoked certificates** that are no longer valid.
- **OCSP:** A protocol used to **check the status of a certificate in real time** instead of downloading an entire CRL.

📌 Chapter 3: How PKI Works

PKI is a **hierarchical model** that ensures **trust and secure communications**.

📌 Steps in PKI Operations:

1. A user requests a certificate from a CA.
2. The CA **verifies the identity** of the applicant.
3. If verification is successful, the CA **issues a digital certificate**.
4. The certificate is **used for encryption, authentication, or digital signing**.
5. The recipient **validates the certificate** before establishing trust.

📌 Chapter 4: Types of Digital Certificates

📌 Common Digital Certificate Types:

Certificate Type	Purpose
SSL/TLS Certificate	Secures website communications via HTTPS
Code Signing Certificate	Ensures software authenticity (e.g., Microsoft updates)
Email Certificate (S/MIME)	Encrypts and signs emails
Client Authentication Certificate	Verifies user identities for VPNs & internal systems

📌 Chapter 5: Digital Signatures & Authentication

5.1 What is a Digital Signature?

A **digital signature** is a **cryptographic technique** that ensures **data integrity and authenticity**.

📌 Steps in Digital Signing:

1. A document is **hashed** (converted into a unique value).
2. The sender **encrypts the hash** using their **private key**.
3. The recipient **decrypts the hash** using the sender's **public key**.
4. If the decrypted hash matches, the document is **authentic and unchanged**.

✓ Uses:

- Signing **PDFs, contracts, and legal documents**.
- Verifying **software authenticity** (Code Signing).

📌 Chapter 6: PKI Challenges & Security Best Practices

6.1 Challenges in PKI Security

1. **Private Key Theft:** If a private key is compromised, attackers can **impersonate a user or organization**.
2. **Expired Certificates:** Expired SSL/TLS certificates cause **website errors and security risks**.
3. **Man-in-the-Middle Attacks (MITM):** Attackers may intercept communications using **fraudulent certificates**.

6.2 PKI Security Best Practices

✓ Use **Hardware Security Modules (HSMs)** to store private keys securely.
✓ Automate **certificate renewals** to prevent expiration issues.
✓ Monitor **certificate revocations** to avoid compromised certificates.

📌 Chapter 7: Case Study – DigiNotar CA Hack (2011)

📌 Attack Overview:

- Hackers **compromised DigiNotar (a CA)** and issued **fraudulent SSL certificates**.
- Users visiting websites were unknowingly being monitored.

📌 Impact:

- Over **500 fraudulent certificates** were issued.
- DigiNotar **lost trust and shut down**.

✓ Lessons Learned:

- Always **use trusted CAs**.
- Implement **certificate pinning** to prevent fake certificate attacks.

📌 Chapter 8: Summary

✅ Key Takeaways

- PKI is **a framework for secure communications and authentication**.

- **Certificate Authorities (CAs) issue digital certificates** to verify identities.
- **Digital signatures ensure data authenticity**.
- **PKI security best practices prevent cyber threats**.

📌 Chapter 9: Next Steps

- Learn **PKI implementation using OpenSSL**.
- Explore **PKI certificate management solutions** (e.g., Let's Encrypt).
- Stay updated on **PKI security threats & countermeasures**.

STEGANOGRAPHY & SECURE COMMUNICATION METHODS

📌 CHAPTER 1: INTRODUCTION TO STEGANOGRAPHY & SECURE COMMUNICATION

1.1 What is Steganography?

Steganography is the **practice of hiding information within digital files or physical objects** to conceal its existence. Unlike encryption, which scrambles data into unreadable forms, steganography ensures that secret information is hidden **inside a cover medium** like an image, audio, video, or text.

🔹 **Key Concept:** Steganography does not change the file's appearance, making it difficult to detect.

📌 **Example of Steganography in Everyday Life:**
✓ **Invisible Watermarks in Images** – Used to protect copyrights.
✓ **Hidden Messages in Digital Images or Audio** – Used for secret communication.
✓ **QR Codes with Embedded Data** – Used in product tracking.

🖼️ **Diagram: Steganography vs. Encryption**

Encryption:

Plain Text ⟶ Encrypted Data (Unreadable) ⟶ Decryption ⟶ Plain Text

Steganography:

Secret Message + Cover Medium ⟶ Stego Object (Looks Normal)

✅ **Key Difference:**

- **Encryption makes data unreadable,** raising suspicion if intercepted.
- **Steganography hides data in plain sight,** making it difficult to detect.

1.2 Importance of Secure Communication

Secure communication is **essential for protecting sensitive information** in various fields like cybersecurity, military intelligence, business transactions, and personal privacy.

🔍 **Why Secure Communication Matters?**

✓ Prevents **unauthorized access** to sensitive information.
✓ Protects **intellectual property** from cybercriminals.
✓ Ensures **privacy in digital messaging**.

📌 **Real-World Example: Edward Snowden Leaks (2013)**

- Snowden used encryption tools and steganography to **safely communicate with journalists** without being detected by intelligence agencies.

✅ **Applications of Secure Communication:**

✓ Secure **email communication** (PGP encryption).
✓ Safe **banking transactions** (TLS, SSL).
✓ **End-to-end encrypted messaging** (Signal, Telegram).

📌 Chapter 2: Types of Steganography

2.1 Image Steganography (Hiding Data in Images)

Image steganography **hides secret messages inside digital images** by modifying pixel values in ways that are **imperceptible to the human eye**.

Methods of Image Steganography:

- 🔹 **Least Significant Bit (LSB) Insertion** – The most common technique that alters the least significant bits of pixel values.
- 🔹 **DCT-based Steganography** – Hides data in the frequency domain of an image (used in JPEG).
- 🔹 **Masking & Filtering** – Hides data in significant image areas, making detection harder.

📌 Example: Hiding a Message in an Image (Using LSB)

1. Convert the message into **binary format**.
2. Modify the **least significant bits** of image pixel values to embed the message.
3. Extract the hidden message from modified pixels.

🖼️ Diagram: LSB Image Steganography

Original Pixel: 10110011

Modified Pixel: 10110010 (Least significant bit changed)

✅ Best Practices:

✓ Use **high-resolution images** to prevent visual detection.

✓ Encrypt the message **before embedding it** for stronger security.

2.2 Audio Steganography (Hiding Data in Sound Files)

Audio steganography **embeds secret messages inside sound files** by altering sound waves.

Methods of Audio Steganography:

- **Echo Hiding** – Introduces tiny echoes that are **inaudible to humans** but contain hidden information.
- **Phase Coding** – Alters the **phase of audio signals** to store secret messages.
- **Spread Spectrum** – Distributes hidden data across multiple frequencies.

📌 **Example: Hiding Data in an MP3 File**

1. Convert text into **binary data**.
2. Modify the **sound wave phase or echo properties** to embed data.
3. Extract the secret message using specialized software.

✅ **Best Practices:**

✓ Use **lossless audio formats** (WAV, FLAC) to avoid data loss.

✓ Avoid **modifying popular music tracks,** as the changes can be detected.

2.3 Text Steganography (Hiding Messages in Text)

Text steganography hides data within text-based documents using **character spacing, font changes, or encoding schemes**.

Methods of Text Steganography:

🔹 **Zero-Width Characters** – Uses **invisible Unicode characters** to hide messages.

🔹 **First Letter Encoding** – Encodes data in **the first letters of words**.

🔹 **Whitespace Encoding** – Uses **extra spaces** at the end of lines to encode binary data.

📌 **Example: Hidden Message Using First Letters**

The quick brown fox jumps over the lazy dog.

Hidden Message: TQBFJOTLD (First letter of each word)

✅ **Best Practices:**

✓ Use **natural-sounding sentences** to avoid suspicion.

✓ Combine **text steganography with encryption** for stronger security.

📌 CHAPTER 3: SECURE COMMUNICATION METHODS

3.1 Cryptography vs. Steganography

Method	Purpose	Example
Encryption	Makes data unreadable without a key	AES, RSA
Steganography	Hides data in normal-looking files	Hiding text in images
Combination	Uses encryption + steganography	Encrypting a message before hiding it

✅ **Best Practice:** Always **encrypt data before hiding it** using steganography.

3.2 Secure Messaging Apps

Secure messaging apps protect conversations from **hacking, interception, and surveillance**.

📌 **Popular Secure Messaging Apps:**

✓ **Signal** – Uses end-to-end encryption and self-destructing messages.

✓ **Telegram (Secret Chats)** – Offers disappearing messages and secure encryption.

✓ **WhatsApp** – Uses the **Signal Protocol** for private messaging.

✅ **Best Practices:**

✓ Avoid using **SMS** or **unencrypted messaging**.

✓ Always verify **encryption keys** before sharing sensitive data.

3.3 Secure File Transfer Methods

Secure file transfer prevents unauthorized access to **documents, images, and sensitive files**.

📌 **Common Secure Transfer Methods:**

✓ **PGP (Pretty Good Privacy)** – Encrypts emails and files with **public-key encryption**.

✓ **SFTP (Secure File Transfer Protocol)** – Encrypts file transfers.

✓ **Onion Routing (Tor Network)** – Hides the source and destination of internet traffic.

✅ **Best Practices:**

✓ Always use **strong encryption algorithms** (AES-256, RSA-4096).

✓ **Never share private encryption keys**.

📌 Chapter 4: Case Study – Steganography in Cybersecurity

📌 Case Study: Terrorists Using Steganography (2011)

- Terrorist organizations used **steganography** to hide secret instructions in images.
- Government agencies developed **AI-powered detection tools** to analyze suspicious images and detect hidden data.

✅ Lesson Learned:

✓ Steganography is used for **both cybersecurity and cybercrime**.
✓ **Detection tools** need to evolve to counter steganographic threats.

📌 Chapter 5: Summary & Next Steps

✅ Key Takeaways:

✓ **Steganography conceals information** inside digital files.
✓ **Different types include image, audio, video, and text steganography**.
✓ **Secure communication tools like Signal, PGP, and Tor** enhance privacy.
✓ **Encryption + Steganography** provides **double-layer security**.

🚀 Next Steps:

🔹 **Practice using Steghide, OpenStego, and StegExpose** to hide & detect messages.
🔹 **Learn forensic techniques** to detect steganography.

- **Explore quantum encryption for future secure communication.**

BLOCKCHAIN & CYBERSECURITY – SECURING TRANSACTIONS

📌 CHAPTER 1: INTRODUCTION TO BLOCKCHAIN & CYBERSECURITY

1.1 What is Blockchain?

Blockchain is a **distributed ledger technology (DLT)** that records digital transactions in a decentralized and tamper-proof manner. Unlike traditional databases controlled by a **central authority**, blockchain operates on a **peer-to-peer network**, where all participants (nodes) have access to a copy of the ledger.

🔹 **Key Features of Blockchain:**

- **Decentralization:** No central control; records are shared across all nodes in the network.
- **Immutability:** Once a transaction is recorded, it cannot be modified or deleted.
- **Transparency:** All transactions are visible and verifiable by network participants.
- **Security:** Uses cryptographic hashing and consensus mechanisms to prevent fraud.

🖼️ **Diagram: How Blockchain Works**

1. A transaction is initiated (e.g., payment transfer).

2. The transaction is verified by nodes (miners/validators).

3. Verified transactions are grouped into a block.

4. The block is added to the blockchain.

5. The transaction is permanently recorded and cannot be changed.

1.2 Why is Blockchain Important for Cybersecurity?

🚨 **Cybersecurity threats in centralized systems include:**

- **Single point of failure:** Centralized databases can be hacked, leading to massive data breaches.
- **Data tampering:** Centralized authorities can modify or delete records.
- **Insider threats:** Employees with privileged access can compromise sensitive information.

🔹 **How Blockchain Enhances Cybersecurity:**

Threat	Blockchain Solution
Data Tampering	Immutable records prevent unauthorized modifications.
Fraudulent Transactions	Consensus mechanisms validate transactions before approval.
Identity Theft	Private-public key encryption secures user identity.
DDoS Attacks	Decentralized network prevents single points of failure.

✓ **Real-World Example:**

- **Bitcoin Blockchain:** Transactions are verified by a decentralized network of nodes, ensuring security and transparency.
- **IBM Hyperledger Fabric:** Provides a permissioned blockchain for enterprises, securing sensitive transactions.

📌 Chapter 2: How Blockchain Secures Transactions

2.1 Cryptographic Techniques Used in Blockchain

Blockchain security relies on **cryptographic algorithms** that ensure confidentiality, integrity, and authentication.

📌 1. Hashing – Ensuring Data Integrity

Hashing is a **one-way function** that converts an input (transaction data) into a fixed-length string.

Example:

Input: "Blockchain Security"

SHA-256 Hash: "b1a88c6f6e518f33c8184f3bc9a..."

✓ **Security Benefit:** If even a single character changes, the hash output changes completely, ensuring data integrity.

📌 2. Public-Key Cryptography (PKC) – Securing Transactions

Blockchain uses **asymmetric encryption** with two keys:

- **Public Key** – Shared with others to receive funds.
- **Private Key** – Kept secret; used to sign transactions.

✓ **Security Benefit:** Only the owner of the **private key** can authorize transactions.

📌 3. Digital Signatures – Authenticating Transactions

Digital signatures verify the sender's identity and ensure that the message or transaction has not been altered.

✓ **Security Benefit:** Prevents **man-in-the-middle attacks,** where hackers try to alter transaction data.

2.2 Consensus Mechanisms – Preventing Fraudulent Transactions

A **consensus mechanism** ensures all nodes in a blockchain network agree on the validity of transactions.

📌 Types of Consensus Mechanisms:

Consensus Mechanism	How It Works	Used In
Proof of Work (PoW)	Miners solve cryptographic puzzles to validate transactions.	Bitcoin, Ethereum (pre-2.0)
Proof of Stake (PoS)	Validators stake coins to confirm transactions.	Ethereum 2.0, Cardano
Delegated Proof of Stake (DPoS)	Users vote for delegates to validate transactions.	EOS, TRON
Byzantine Fault Tolerance (BFT)	Nodes reach agreement despite potential malicious actors.	Hyperledger Fabric

✓ Security Benefit:

- Prevents **double-spending attacks,** where a user tries to spend the same digital asset twice.

📌 Chapter 3: Blockchain Applications in Cybersecurity

3.1 Securing Identity and Authentication

📌 How Blockchain Protects Identities:

- Users control their data using **decentralized identity management**.
- Eliminates reliance on **centralized databases** (which are prone to hacking).
- Digital signatures ensure **identity authentication** without passwords.

✓ Real-World Example:

- **Microsoft's ION** – A decentralized identity system that secures user identities without relying on traditional authentication methods.

3.2 Preventing Data Tampering & Supply Chain Security

📌 Blockchain Benefits for Data Security:

- **Tamper-proof storage** ensures records cannot be altered.
- **Supply chain tracking** allows transparent monitoring of transactions.

✓ Real-World Example:

- **IBM Food Trust** – Uses blockchain to track food supply chains, ensuring security and authenticity.

3.3 Smart Contracts – Automating Secure Transactions

📌 What are Smart Contracts?

Smart contracts are **self-executing agreements** where conditions are coded into blockchain networks.

✓ Security Benefit:

- **Eliminates intermediaries,** reducing risks of fraud.
- **Automatic execution** ensures that contractual obligations are met.

📌 Example of a Smart Contract in Solidity (Ethereum's Programming Language):

```
pragma solidity ^0.8.0;

contract PaymentContract {

  address payable recipient;

  function sendPayment() public payable {

    recipient.transfer(msg.value);

  }

}
```

✓ Real-World Example:

- **Ethereum's DeFi Platforms** – Secure financial transactions using smart contracts.

📌 Chapter 4: Cybersecurity Risks in Blockchain

4.1 Blockchain Security Vulnerabilities

Threat	How It Works
51% Attack	A group controls most of the network's hashing power, altering transactions.
Private Key Theft	If a private key is stolen, attackers gain full control of assets.
Smart Contract Bugs	Vulnerabilities in smart contract code can be exploited.
Phishing Attacks	Attackers trick users into revealing private keys.

✓ Best Practices:

- **Use hardware wallets** to protect private keys.
- **Audit smart contracts** before deployment.
- **Enable multi-signature transactions** for enhanced security.

📌 Chapter 5: Case Study – The Mt. Gox Bitcoin Hack (2014)

📌 Attack Overview:

- Hackers exploited vulnerabilities in **Mt. Gox's centralized exchange**, stealing **850,000 Bitcoins** (~$450 million at the time).

📌 Impact:

- Investors lost funds permanently due to the **lack of blockchain security protocols**.
- Highlighted the need for **decentralized exchanges** and **cold storage wallets**.

✓ **Lessons Learned:**

- Never store large amounts of cryptocurrency in **centralized exchanges**.
- Use **cold wallets (offline storage)** for enhanced security.

📌 Chapter 6: Summary

✅ **Key Takeaways:**

- **Blockchain enhances cybersecurity** through decentralization, encryption, and smart contracts.
- **Consensus mechanisms prevent fraud** by ensuring trustless transactions.
- **Smart contracts automate security processes** but require proper auditing.
- **Cyber threats like 51% attacks and private key theft must be mitigated**.

📌 Chapter 7: Next Steps

- **Learn about blockchain security tools** like MetaMask, Ledger, and Trezor.
- **Explore advanced blockchain topics** like Zero-Knowledge Proofs (ZKP) and Secure Multi-Party Computation (SMPC).
- **Practice developing secure smart contracts** on Ethereum's test network.

ASSIGNMENT: ENCRYPTING & SECURING DATA

TASK: IMPLEMENT AES/RSA ENCRYPTION FOR SECURING SENSITIVE DATA.

OBJECTIVE: UNDERSTAND HOW CRYPTOGRAPHY IS USED TO PROTECT INFORMATION.

📝 Assignment: Encrypting & Securing Data

📌 Task: Implement AES/RSA Encryption for Securing Sensitive Data

📌 Objective: Understand How Cryptography is Used to Protect Information

---

📌 Step 1: Understanding AES and RSA Encryption

Before implementing encryption, it's important to understand the **difference between AES and RSA** and when to use them.

1.1 AES (Advanced Encryption Standard) – Symmetric Encryption

✔ **Uses a single key** for both encryption and decryption.
✔ **Fast and efficient** – suitable for encrypting large amounts of data.
✔ **Common Use Cases:** File encryption, secure messaging, Wi-Fi security.

🖼️ How AES Works:

Plaintext → AES Key → Encryption → Ciphertext

Ciphertext → AES Key → Decryption → Plaintext

1.2 RSA (Rivest-Shamir-Adleman) – Asymmetric Encryption

✔ Uses **two keys**: **Public Key (encryption)** and **Private Key (decryption)**.
✔ **Slower than AES** – used mainly for secure key exchange and authentication.
✔ **Common Use Cases:** Secure emails, SSL/TLS (web encryption), digital signatures.

🖼️ **How RSA Works:**

Plaintext → Public Key (Encryption) → Ciphertext

Ciphertext → Private Key (Decryption) → Plaintext

✔ **Best Practice:** Use **AES for encrypting large data** and **RSA to encrypt the AES key** (Hybrid Encryption).

📌 Step 2: Setting Up the Environment

Before implementing encryption, install the necessary cryptographic libraries.

✔ **Required Libraries:**

- **PyCryptodome** (for AES & RSA encryption in Python)
- **base64** (for encoding encrypted data)

📌 **Install PyCryptodome:**

pip install pycryptodome

📌 Step 3: Implementing AES Encryption (Symmetric Encryption)

3.1 Generating an AES Key

AES requires a **secret key** that must be **securely stored**.

```
from Crypto.Random import get_random_bytes

# Generate a 256-bit AES key
aes_key = get_random_bytes(32)  # 32 bytes = 256 bits
print("AES Key:", aes_key.hex())  # Convert to hex for storage
```

3.2 Encrypting Data Using AES

We will use **AES in EAX mode** for authenticated encryption.

```
from Crypto.Cipher import AES
import base64

def aes_encrypt(plaintext, key):
    cipher = AES.new(key, AES.MODE_EAX)  # Create AES cipher
    ciphertext, tag = cipher.encrypt_and_digest(plaintext.encode())  # Encrypt message
    return base64.b64encode(cipher.nonce + tag + ciphertext).decode()  # Encode in Base64
```

```
# Example usage
plaintext = "Confidential Data"
encrypted_text = aes_encrypt(plaintext, aes_key)
print("AES Encrypted:", encrypted_text)
```

✔ **Explanation:**

- The **nonce** ensures uniqueness for each encryption.
- The **tag** is used for message integrity.
- **Base64 encoding** makes it easier to store and transmit the ciphertext.

3.3 Decrypting AES Encrypted Data

To retrieve the original message, decrypt the ciphertext using the same AES key.

```
def aes_decrypt(encrypted_text, key):
    raw_data = base64.b64decode(encrypted_text)  # Decode from Base64
    nonce, tag, ciphertext = raw_data[:16], raw_data[16:32], raw_data[32:]  # Extract components
    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    return cipher.decrypt_and_verify(ciphertext, tag).decode()
```

```
# Example usage
decrypted_text = aes_decrypt(encrypted_text, aes_key)
print("AES Decrypted:", decrypted_text)
```

✔ **Key Points:**

- The **nonce and tag** ensure that any tampering is detected.
- The AES **key must be kept secret** to prevent unauthorized decryption.

📌 Step 4: Implementing RSA Encryption (Asymmetric Encryption)

RSA uses a **public-private key pair** for encryption and decryption.

4.1 Generating RSA Keys

```
from Crypto.PublicKey import RSA

# Generate RSA key pair
key = RSA.generate(2048)
private_key = key.export_key()  # Private key (keep secure)
public_key = key.publickey().export_key()  # Public key (can be shared)

# Save keys to files
```

```
with open("rsa_private.pem", "wb") as priv_file:
    priv_file.write(private_key)

with open("rsa_public.pem", "wb") as pub_file:
    pub_file.write(public_key)

print("RSA Keys Generated Successfully")
```

- **Private Key**: Used for decryption (keep secure).
- **Public Key**: Used for encryption (can be shared).

4.2 Encrypting Data Using RSA

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
import base64

def rsa_encrypt(plaintext, public_key_file):
    public_key = RSA.import_key(open(public_key_file).read())  # Load public key
    cipher = PKCS1_OAEP.new(public_key)  # Create RSA cipher
    ciphertext = cipher.encrypt(plaintext.encode())  # Encrypt message
```

```
    return base64.b64encode(ciphertext).decode()  # Encode in Base64


# Example usage
plaintext = "Confidential Data"
encrypted_rsa = rsa_encrypt(plaintext, "rsa_public.pem")
print("RSA Encrypted:", encrypted_rsa)
```

✔ **Why Use Base64?**

RSA ciphertext is **binary data**; encoding in **Base64** makes it **easier to store and transmit**.

4.3 Decrypting RSA Encrypted Data

```
def rsa_decrypt(encrypted_text, private_key_file):
    private_key = RSA.import_key(open(private_key_file).read())  # Load private key
    cipher = PKCS1_OAEP.new(private_key)
    decrypted_text = cipher.decrypt(base64.b64decode(encrypted_text))  # Decrypt
    return decrypted_text.decode()


# Example usage
decrypted_rsa = rsa_decrypt(encrypted_rsa, "rsa_private.pem")
print("RSA Decrypted:", decrypted_rsa)
```

✔ Why is RSA Secure?

- **2048-bit RSA keys** provide strong security.
- Even with modern computing power, **breaking RSA would take years**.

📌 Step 5: Combining AES and RSA (Hybrid Encryption)

AES is **fast but requires a secret key**, while RSA is **secure but slow for large data**. The best approach is to **encrypt data with AES** and **encrypt the AES key using RSA**.

5.1 Encrypting the AES Key with RSA

```
def encrypt_aes_key(aes_key, public_key_file):
    return rsa_encrypt(aes_key.hex(), public_key_file)  # Convert AES key to hex before encryption


encrypted_aes_key = encrypt_aes_key(aes_key, "rsa_public.pem")
print("Encrypted AES Key:", encrypted_aes_key)
```

✔ Why Encrypt the AES Key?

- Instead of sharing the AES key directly, **we encrypt it using RSA**.
- This ensures that only the **intended recipient with the RSA private key** can decrypt the AES key.

5.2 Decrypting the AES Key with RSA

```
def decrypt_aes_key(encrypted_aes_key, private_key_file):
    hex_key = rsa_decrypt(encrypted_aes_key, private_key_file)  # Decrypt AES key
    return bytes.fromhex(hex_key)  # Convert back to bytes


decrypted_aes_key = decrypt_aes_key(encrypted_aes_key, "rsa_private.pem")
print("Decrypted AES Key:", decrypted_aes_key)
```

✔ **Final Process:**

1. Sender **encrypts data with AES**.
2. AES **key is encrypted with RSA public key**.
3. Receiver **decrypts AES key using RSA private key**.
4. Receiver **uses decrypted AES key to decrypt data**.

📌 CONCLUSION

✅ **What You Learned**

- **AES** is used for **fast, secure encryption** of large data.
- **RSA** is used for **secure key exchange and digital signatures**.
- **Hybrid encryption** (AES + RSA) combines **speed and security**.

🚀 Next Steps

- Learn **Elliptic Curve Cryptography (ECC)** for lightweight encryption.
- Implement **Digital Signatures** for authentication.
- Experiment with **cryptographic attacks and defenses**.
