Blockchain L01

charudatta

Contents

Introduction to Cryptography Concepts 1
Cryptography Basics 2
Encryption and Decryption 2
Symmetric Encryption 2
Asymmetric Encryption 2
Symmetric vs Asymmetric Encryption 2
Real-World Applications 3
Asymmetric Encryption: SSL/TLS for secure web browsing, email encryption, digital signatures. 3
Introduction to Cryptography Concepts 3
Hash Function 3
Hash Pointers 3
One-Way Functions 4
Cryptographic Protocols: Fundamental in public key cryptography and digital signatures. 4
Introduction to Encryption 4
AES (Advanced Encryption Standard) 5
AES Structure 5
DES (Data Encryption Standard) 5
3DES (Triple DES) 5
RSA (Rivest–Shamir–Adleman) 5
ECC (Elliptic Curve Cryptography) 6
DSA (Digital Signature Algorithm) 6
Used in digital authentication. 6
Smart Contract: A self-executing contract with the terms of the agreement directly written into code. 8
Fallback and Receive Functions: Handle Ether transactions. 11

Introduction to Cryptography Concepts

Cryptography Basics

- **Definition**: The art of securing communication through encoding information.
- **Objective**: Ensure confidentiality, integrity, and authenticity of data.

Encryption and Decryption

- **Encryption**: Process of converting plaintext into ciphertext.
- **Decryption**: Process of converting ciphertext back into plaintext.
- **Purpose**: Protect data from unauthorized access.

Symmetric Encryption

- **Definition**: Uses the same key for both encryption and decryption.
- **Examples**: AES, DES, 3DES
- **Pros**:
  - Fast and efficient.
  - Suitable for large data encryption.
- **Cons**:
  - Key distribution can be challenging.
  - Compromised key means compromised data.

Asymmetric Encryption

- **Definition**: Uses a pair of keys—public key for encryption and private key for decryption.
- **Examples**: RSA, ECC, DSA
- **Pros**:
  - Simplifies key distribution.
  - Provides digital signatures.
- **Cons**:
  - Slower than symmetric encryption.
  - Computationally intensive.

Symmetric vs Asymmetric Encryption

Feature	Symmetric Encryption	Asymmetric Encryption
Key Usage	Same key for encryption/decryption	Public and private key pairs
Speed	Faster	Slower
Key Distribution	Challenging	Easier (public key can be shared)
Data Security	Compromised if key is exposed	More secure even if public key is known

---

Real-World Applications

- **Symmetric Encryption**: Securing data at rest, VPNs, disk encryption.
-
  ## Asymmetric Encryption: SSL/TLS for secure web browsing, email encryption, digital signatures.

  marp: true theme: default paginate: true class: invert title: "Blockchain L02: Hash" author: charudatta created: 22-10-20024 header: 'Blockchain Analyses Class' footer: 'Unit I: Cryptography - By Charudatta korde' tags: nfsu, notes, ppt, blockchain —

Introduction to Cryptography Concepts

Hash Function

- **Definition**: A function that converts input data into a fixed-size string of characters, typically a hash code.
- **Properties**:
  - Deterministic: Same input produces the same output.
  - Fast Computation: Quickly computes the hash value.
  - Pre-image Resistance: Hard to reverse-engineer the input from the hash.
  - Small Change in Input: Produces vastly different hash output.
- **Examples**: MD5, SHA-1, SHA-256

Hash Pointers

- **Definition**: A pointer to where information is stored, along with a cryptographic hash of the information.

- **Uses**:
  - Blockchains: Linking blocks in a chain where each block contains a hash pointer to the previous block.
  - Tamper Detection: Any alteration in the data changes the hash, revealing tampering.
- **Benefits**:
  - Ensures data integrity.
  - Facilitates secure data linking and chain verification.

One-Way Functions

- **Definition**: Functions that are easy to compute in one direction but hard to reverse compute.
- **Properties**:
  - Easy to Compute: Efficient in computing the forward direction.
  - Hard to Invert: Infeasible to determine the input from the output.
- **Examples**:
  - Hash Functions: As discussed, they are a type of one-way function.
  - Mathematical Problems: Certain problems, like factoring large prime products, act as one-way functions.
- **Applications**:
  - Password Storage: Storing hashed passwords.
  -

    ## Cryptographic Protocols: Fundamental in public key cryptography and digital signatures.

    marp: true theme: default class: lead paginate: true title: "Blockchain L03" author: charudatta created: 22-10-20024 header: 'Blockchain Analyses Class' footer: 'Unit I: Cryptography - By Charudatta korde' tags: nfsu, notes, ppt, blockchain —

Introduction to Encryption

Definition: Encryption is the process of converting plaintext into ciphertext to protect data from unauthorized access.

Types:

- Symmetric Encryption
- Asymmetric Encryption

AES (Advanced Encryption Standard)

Overview:

- A symmetric encryption algorithm standardized by NIST.
- Key Sizes: 128, 192, and 256 bits.
- Rounds: 10, 12, or 14 based on key size.

Applications:

- Used in government and commercial systems.

AES Structure

DES (Data Encryption Standard)

Overview:

- An older symmetric encryption algorithm standardized by NIST.
- Key Size: 56 bits.
- Rounds: 16 rounds of processing.

Applications:

- Used in legacy systems and industries.

3DES (Triple DES)

Overview:

- An enhancement of DES to increase security.
- Key Size: Effective key size of 112 or 168 bits.
- Process: Encrypt-Decrypt-Encrypt (EDE) with different keys.

Applications:

- Used in financial systems.

RSA (Rivest–Shamir–Adleman)

Overview:

- An asymmetric encryption algorithm based on the difficulty of factoring large integers.
- Key Sizes: 1024, 2048, or 4096 bits.

- Public and Private Keys: Used for encryption and decryption.

Applications:

- Used in secure communications and digital signatures.

ECC (Elliptic Curve Cryptography)

Overview:

- An asymmetric encryption algorithm based on elliptic curves.
- Key Sizes: Provides similar security to RSA with smaller key sizes.

Applications:

- Used in mobile devices and secure communications.

DSA (Digital Signature Algorithm)

Overview:

- A standard for digital signatures.
- Key Sizes: 1024, 2048, or 3072 bits.
- Process: Involves key generation, signature generation, and signature verification.

Applications:

-

  ### Used in digital authentication.

  title: "07_smartcontract" time: "2025-01-16 :: 22:36:30" tags: nfsu, notes, blockchain —

1. Setting Up Ganache Ganache is a personal blockchain for Ethereum development that you can use to deploy contracts, develop applications, and run tests.

1. **Download and Install Ganache**: Visit the Ganache website and download the appropriate version for your operating system.
2. **Start Ganache**: Open Ganache and start a new blockchain instance. You'll see a screen with accounts, balances, and other information.
3. **Note the RPC Server Address**: This address will be used to connect Remix to your local blockchain.

2. Setting Up Remix IDE Remix is an online IDE for Solidity, the programming language used to write smart contracts on Ethereum.

1. **Access Remix**: Go to the Remix IDE website (https://remix.ethereum.org/).
2. **Create a New File**: In Remix, create a new Solidity file by clicking on the "+" icon and selecting "Solidity" from the dropdown menu.

3. Writing a Smart Contract Let's write a simple smart contract that performs basic arithmetic operations (addition and subtraction).

```
pragma solidity ^0.8.0;

contract Calculator {
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }

    function subtract(uint256 a, uint256 b) public pure returns (uint256) {
        return a - b;
    }
}
```

4. Compiling the Smart Contract

1. **Select the Environment**: In Remix, select the "Solidity Compiler" plugin from the left sidebar.
2. **Compile**: Click the "Compile" button to compile your smart contract.

5. Deploying the Smart Contract

1. **Select the Environment**: In Remix, select the "Deploy & Run Transactions" plugin from the left sidebar.
2. **Deploy**: Click the "Deploy" button and select the environment as "Injected Web3" (which connects to Ganache).
3. **Confirm Deployment**: Confirm the deployment transaction in Ganache and Remix.

6. Interacting with the Smart Contract

1. **Call Functions**: After deployment, you can interact with your smart contract by calling its functions.
2. **Check Transactions**: Use Ganache's built-in block explorer to view the transactions and state changes.

Summary

- **Ganache**: Provides a local blockchain instance for development and testing.
- **Remix**: An online IDE for writing and deploying Solidity smart contracts.
-

  ## Smart Contract: A self-executing contract with the terms of the agreement directly written into code.

  title: "08_smartcontracts" time: "2025-01-16 :: 22:39:42" tags: nfsu, notes, blockchain —

Key Components of a Solidity Smart Contract

1. Pragma Directive The pragma directive specifies the Solidity compiler version to use. It's essential to ensure compatibility and prevent issues due to version changes.

```
pragma solidity ^0.8.0;
```

2. Contract Declaration A contract in Solidity is similar to a class in object-oriented programming. It encapsulates data and functions that operate on that data.

```
contract MyContract {
    // Contract code goes here
}
```

3. State Variables State variables are stored on the blockchain and maintain their values between function calls.

```
contract MyContract {
    uint256 public myNumber;
    string private myString;
}
```

4. Functions Functions are executable units of code within a contract. They can be public, private, internal, or external.

```
contract MyContract {
    uint256 public myNumber;

    // Public function
    function setNumber(uint256 _number) public {
        myNumber = _number;
    }
```

```
    // Private function
    function _privateFunction() private {
        // Private logic
    }

    // Internal function
    function _internalFunction() internal {
        // Internal logic
    }

    // External function
    function externalFunction() external view returns (uint256) {
        return myNumber;
    }
}
```

5. Modifiers Modifiers are used to change the behavior of functions in a declarative way. They are often used for access control.

```
contract MyContract {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not the contract owner");
        _;
    }

    function setNumber(uint256 _number) public onlyOwner {
        // Function logic
    }
}
```

6. Events Events allow contracts to log information that can be listened to by external applications (e.g., DApps) or scripts.

```
contract MyContract {
    event NumberSet(uint256 indexed number);

    function setNumber(uint256 _number) public {
        emit NumberSet(_number);
    }
```

```
}
```

7. Constructor The constructor is an optional function that is executed once when the contract is deployed. It's typically used for initializing state variables.

```
contract MyContract {
    uint256 public myNumber;

    constructor(uint256 _initialNumber) {
        myNumber = _initialNumber;
    }
}
```

8. Fallback and Receive Functions These functions handle Ether transactions sent to the contract.

- **receive**: Handles plain Ether transfers.
- **fallback**: Handles calls with data or those that don't match any function signature.

```
contract MyContract {
    event Received(address sender, uint256 amount);

    receive() external payable {
        emit Received(msg.sender, msg.value);
    }

    fallback() external payable {
        // Fallback logic
    }
}
```

Detailed Example

Here's a complete example incorporating all these elements:

```
pragma solidity ^0.8.0;

contract MyContract {
    uint256 public myNumber;
    address public owner;

    event NumberSet(uint256 indexed number);
    event Received(address sender, uint256 amount);

    constructor(uint256 _initialNumber) {
        owner = msg.sender;
        myNumber = _initialNumber;
```

```
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not the contract owner");
        _;
    }

    function setNumber(uint256 _number) public onlyOwner {
        myNumber = _number;
        emit NumberSet(_number);
    }

    function getNumber() public view returns (uint256) {
        return myNumber;
    }

    receive() external payable {
        emit Received(msg.sender, msg.value);
    }

    fallback() external payable {
        // Fallback logic
    }
}
```

Summary

- **Pragma Directive**: Specifies the compiler version.
- **Contract Declaration**: Defines the contract.
- **State Variables**: Store persistent data.
- **Functions**: Contain executable code.
- **Modifiers**: Modify function behavior.
- **Events**: Log information.
- **Constructor**: Initializes the contract.
-

  ## Fallback and Receive Functions: Handle Ether transactions.

  title: "09_summary_notes.md" time: "2025-02-05 :: 15:10:44" tags: nfsu, notes, blockchain —

Introduction to Cryptography Concepts

---

1. Encryption and Decryption

- **Encryption**: Process of converting plaintext into ciphertext using an algorithm and a key.
- **Decryption**: Reverse process of converting ciphertext back to plaintext.
- **Types of Encryption**:
  - **Symmetric Encryption**:
    - Uses the same key for encryption and decryption.
    - Examples: AES, DES.
    - Pros: Fast and efficient.
    - Cons: Key distribution is challenging.
  - **Asymmetric Encryption**:
    - Uses a pair of keys: public key (encryption) and private key (decryption).
    - Examples: RSA, ECC.
    - Pros: Solves key distribution problem.
    - Cons: Slower than symmetric encryption.

---

2. Hash Functions, Hash Pointers, and One-Way Functions

- **Hash Function**:
  - Takes input data and produces a fixed-size output (hash).
  - Properties: Deterministic, fast computation, pre-image resistance, collision resistance.
  - Examples: SHA-256, Keccak.
- **Hash Pointers**:
  - A pointer to data along with its hash value.
  - Used in blockchain to ensure data integrity.
- **One-Way Functions**:
  - Easy to compute in one direction but hard to reverse.
  - Fundamental to cryptographic security.

---

3. Digital Signatures — ECDSA

- **Digital Signatures**:
  - Used to verify the authenticity and integrity of a message.
  - Created using the sender's private key and verified using their public key.
- **ECDSA (Elliptic Curve Digital Signature Algorithm)**:
  - A type of digital signature based on elliptic curve cryptography.

- More efficient than RSA in terms of key size and computational requirements.

---

4. **Memory Hard Algorithms & Zero-Knowledge Proofs**

- **Memory Hard Algorithms**:
  - Designed to require significant memory to compute, making them resistant to hardware-based attacks.
  - Example: Scrypt (used in Litecoin).
- **Zero-Knowledge Proofs (ZKP)**:
  - Allows one party to prove knowledge of a secret without revealing the secret itself.
  - Example: zk-SNARKs (used in Zcash).

---

5. **Byzantine General Problem and Fault Tolerance**

- **Byzantine General Problem**:
  - A problem in distributed systems where participants must agree on a strategy despite malicious actors.
- **Fault Tolerance**:
  - The ability of a system to continue functioning even if some components fail.
  - Solved in blockchain through consensus algorithms like Proof of Work (PoW) and Proof of Stake (PoS).

---

6. **Introduction to Quantum Computing**

- **Quantum Computing**:
  - Uses quantum bits (qubits) to perform computations.
  - Can solve certain problems exponentially faster than classical computers.
- **Challenges to Classical Cryptography**:
  - Quantum computers can break asymmetric encryption algorithms (e.g., RSA, ECC) using Shor's algorithm.
  - Symmetric encryption and hash functions are less vulnerable but may require larger key sizes.

---

7. **Blockchain Introduction**

- **Blockchain**:
  - A decentralized, distributed ledger technology.

- Records transactions in blocks linked using cryptographic hashes.

- **Comparison with Conventional Databases**:
  - Blockchain: Immutable, decentralized, transparent.
  - Conventional Databases: Centralized, mutable, controlled by a single entity.

---

8. Blockchain Network and Mining Mechanism

- **Blockchain Network**:
  - Composed of nodes that validate and propagate transactions.
- **Mining Mechanism**:
  - Process of adding new blocks to the blockchain.
  - Miners solve cryptographic puzzles (Proof of Work) to validate transactions.

---

9. Distributed Consensus

- **Consensus Algorithms**:
  - Ensure all nodes agree on the state of the blockchain.
  - Examples: Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS).

---

10. Merkle Patricia Tree, Transactions, and Fees

- **Merkle Patricia Tree**:
  - A data structure used in Ethereum to efficiently store and verify large datasets.
- **Transactions**:
  - Actions initiated by users to transfer assets or execute smart contracts.
- **Fees**:
  - Paid to miners/validators for processing transactions.

---

11. Anonymity, Reward, and Chain Policy

- **Anonymity**:
  - Achieved through techniques like ring signatures (Monero) or zero-knowledge proofs (Zcash).
- **Reward**:
  - Incentives given to miners/validators for securing the network.
- **Chain Policy**:
  - Rules governing the blockchain (e.g., block size, block time).

---

12. Life of a Blockchain Application

- Steps:
  1. Ideation and use case identification.
  2. Protocol design and consensus mechanism selection.
  3. Development and testing.
  4. Deployment and network launch.
  5. Maintenance and upgrades (e.g., forks).

---

13. Soft and Hard Forks, Private and Public Blockchain

- **Soft Fork**:
  - Backward-compatible upgrade to the blockchain.
- **Hard Fork**:
  - Non-backward-compatible upgrade, often resulting in a new chain.
- **Private Blockchain**:
  - Restricted access, controlled by a single organization.
- **Public Blockchain**:
  - Open to anyone, fully decentralized.

---

14. Nakamoto Consensus, Proof of Work (PoW)

- **Nakamoto Consensus**:
  - The consensus mechanism used in Bitcoin.
  - Relies on Proof of Work to achieve consensus.
- **Proof of Work (PoW)**:
  - Miners solve computational puzzles to validate transactions and create new blocks.

---

15. Proof of Stake (PoS), Proof of Burn (PoB)

- **Proof of Stake (PoS)**:
  - Validators are chosen based on the number of tokens they hold and are willing to "stake."
  - More energy-efficient than PoW.
- **Proof of Burn (PoB)**:
  - Validators "burn" tokens to gain the right to validate transactions.

---

16. Difficulty Level, Sybil Attack, Energy Utilization

- **Difficulty Level**:
  - Adjusts to maintain a consistent block creation time (e.g., 10 minutes in Bitcoin).
- **Sybil Attack**:
  - An attack where a single entity creates multiple fake identities to gain control over the network.
- **Energy Utilization**:
  - PoW blockchains (e.g., Bitcoin) consume significant energy, leading to environmental concerns.

---

17. Alternate Smart Contract Construction

- **Smart Contracts**:
  - Self-executing contracts with terms directly written into code.
- **Alternate Constructions**:
  - Platforms like Ethereum, Solana, and Cardano offer different approaches to smart contract execution.