**Blockchain Programs**

**Name: Ajay Jayaprakash**

**USN : 1MS21CS009**

**Program 1**

import java.util.ArrayList;

import java.util.List;

import java.security.MessageDigest;

class DataNode {

int marks;

String hashValue;

DataNode previousNode;

String previousHashValue;

public DataNode(int marks, DataNode previousNode) {

this.marks = marks;

this.previousNode = previousNode;

this.previousHashValue = previousNode != null ? previousNode.hashValue : "0";

this.hashValue = calculateHash();

}

private String calculateHash() {

try {

MessageDigest digest = MessageDigest.getInstance("SHA-256");

String data = marks + previousHashValue;

byte[] hash = digest.digest(data.getBytes());

StringBuilder hexString = new StringBuilder();

for (byte b : hash) {

String hex = Integer.toHexString(0xff & b);

if (hex.length() == 1) hexString.append('0');

hexString.append(hex);

}

return hexString.toString();

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

}

public class Blockchain {

private List<DataNode> nodeList;

public Blockchain() {

this.nodeList = new ArrayList<>();

}

public void addNode(int marks) {

DataNode previousNode = nodeList.isEmpty() ? null : nodeList.get(nodeList.size() - 1);

DataNode newNode = new DataNode(marks, previousNode);

nodeList.add(newNode);

}

public void printBlockchain() {

int i = 1;

for (DataNode node : nodeList) {

System.out.println("Block " + i + "\n\tMarks: " + node.marks + "\n\tHash Value: " + node.hashValue + "\n\tPrevious Hash Value: " + node.previousHashValue + "\n\n");

i = i + 1;

}

}

public static void main(String[] args) {

Blockchain block = new Blockchain();

block.addNode(90);

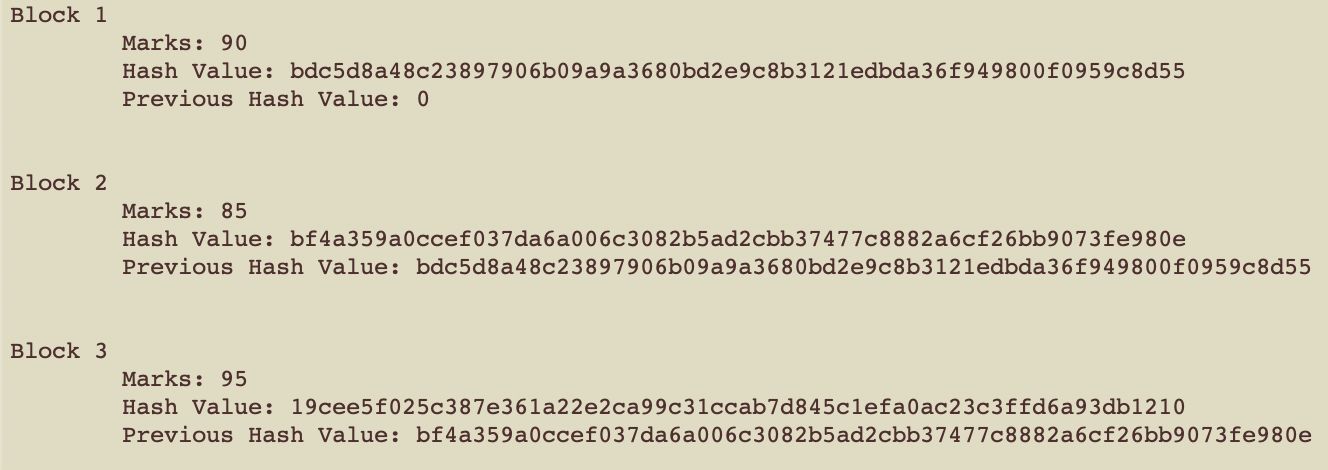
block.addNode(85);

block.addNode(95);

block.printBlockchain();

}

}



**Program 2**

**Client**

import java.io.\*;

import java.net.\*;

public class BlockchainClient {

private static final String SERVER\_IP = "localhost";

private static final int PORT = 12345;

public static void main(String[] args) {

try {

Socket socket = new Socket(SERVER\_IP, PORT);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

String serverResponse;

while ((serverResponse = in.readLine()) != null) {

System.out.println(serverResponse);

if (serverResponse.contains("Enter username:") || serverResponse.contains("Enter password:")

|| serverResponse.contains("Enter the number of bitcoins to buy (or 'exit' to quit):")) {

String userInputStr = userInput.readLine();

out.println(userInputStr);

if (userInputStr.equalsIgnoreCase("exit")) {

break;

}

}

}

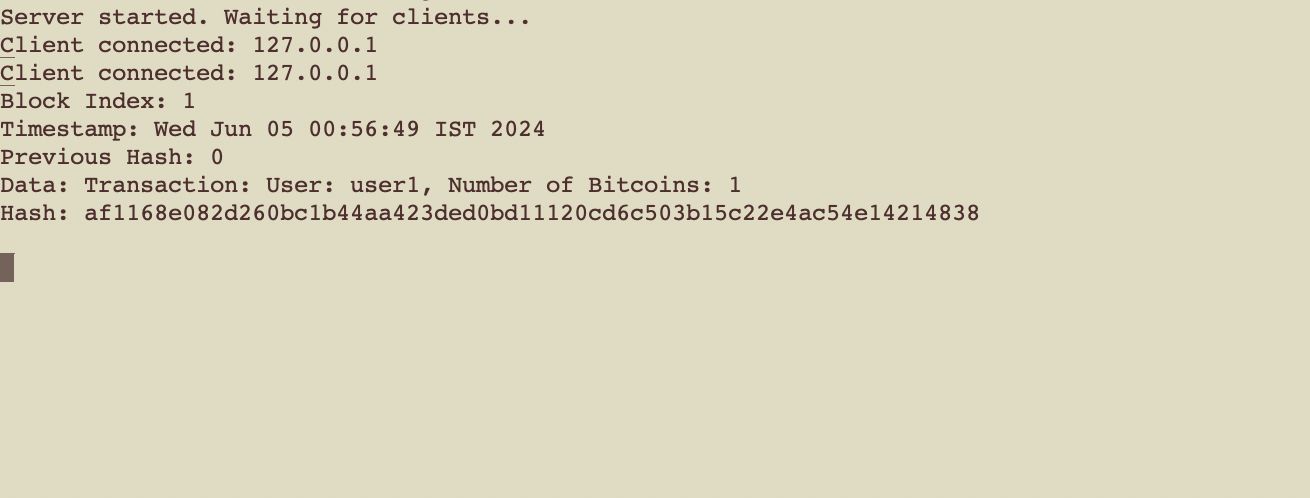
socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Server**

import java.io.\*;

import java.net.\*;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.\*;

public class BlockchainServer {

private static final int PORT = 12345;

private static final double BITCOIN\_RATE = 5500000; // 55 lakhs

private static class Block {

private int index;

private String timestamp;

private String previousHash;

private String data;

private String hash;

public Block(int index, String timestamp, String previousHash, String data, String hash) {

this.index = index;

this.timestamp = timestamp;

this.previousHash = previousHash;

this.data = data;

this.hash = hash;

}

// Getters and setters

}

private static List<Block> blockchain = new ArrayList<>();

private static Object blockchainLock = new Object();

private static Map<String, String> credentials = new HashMap<>(); // Map to store username and password

private static Map<String, Double> balances = new HashMap<>(); // Map to store balances

private static Map<String, Integer> bitcoinsOwned = new HashMap<>(); // Map to store number of bitcoins owned

static {

// Initialize some example credentials, balances, and bitcoinsOwned

credentials.put("user1", "password1");

credentials.put("user2", "password2");

balances.put("user1", 10000000.0);

balances.put("user2", 20000000.0);

bitcoinsOwned.put("user1", 0);

bitcoinsOwned.put("user2", 0);

}

private static class ClientHandler extends Thread {

private Socket clientSocket;

private BufferedReader in;

private PrintWriter out;

private String username;

private String password;

private double accountBalance;

private int numberOfBitcoins;

public ClientHandler(Socket socket) {

this.clientSocket = socket;

}

@Override

public void run() {

try {

in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

out = new PrintWriter(clientSocket.getOutputStream(), true);

// Read username and password

out.println("Enter username:");

username = in.readLine();

out.println("Enter password:");

password = in.readLine();

// Perform authentication

if (authenticate(username, password)) {

out.println("Authentication successful.");

handleTransaction();

} else {

out.println("Error: Authentication failed. Closing connection.");

clientSocket.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private boolean authenticate(String username, String password) {

// Check if username and password match in the credentials map

return credentials.containsKey(username) && credentials.get(username).equals(password);

}

private void handleTransaction() throws IOException {

accountBalance = balances.get(username);

while (true) {

out.println("Bitcoin rate: " + BITCOIN\_RATE);

out.println("Your balance: " + accountBalance);

out.println("Enter the number of bitcoins to buy (or 'exit' to quit):");

String input = in.readLine();

if (input.equalsIgnoreCase("exit")) {

break;

}

try {

int bitcoinsToBuy = Integer.parseInt(input);

double amountToDeduct = bitcoinsToBuy \* BITCOIN\_RATE;

if (accountBalance >= amountToDeduct) {

accountBalance -= amountToDeduct;

numberOfBitcoins += bitcoinsToBuy;

// Update balances and bitcoinsOwned

balances.put(username, accountBalance);

bitcoinsOwned.put(username, numberOfBitcoins);

// Create a new block for this transaction

String currentTimestamp = new Date().toString();

String previousHash = blockchain.isEmpty() ? "0" : blockchain.get(blockchain.size() - 1).hash;

String data = "Transaction: User: " + username + ", Number of Bitcoins: " + bitcoinsToBuy;

String currentHash = computeHash(currentTimestamp, previousHash, data);

Block block = new Block(blockchain.size() + 1, currentTimestamp, previousHash, data, currentHash);

synchronized (blockchainLock) {

blockchain.add(block);

}

out.println("Transaction successful. You now own " + numberOfBitcoins + " bitcoins.");

out.println("Current hash: " + currentHash);

displayBlockchain();

} else {

out.println("Error: Insufficient balance.");

}

} catch (NumberFormatException e) {

out.println("Error: Invalid input.");

}

}

clientSocket.close();

}

private void displayBlockchain() {

synchronized (blockchainLock) {

for (Block block : blockchain) {

System.out.println("Block Index: " + block.index);

System.out.println("Timestamp: " + block.timestamp);

System.out.println("Previous Hash: " + block.previousHash);

System.out.println("Data: " + block.data);

System.out.println("Hash: " + block.hash);

System.out.println();

}

}

}

private String computeHash(String timestamp, String previousHash, String data) {

String dataToHash = timestamp + previousHash + data;

try {

MessageDigest digest = MessageDigest.getInstance("SHA-256");

byte[] hashBytes = digest.digest(dataToHash.getBytes());

StringBuilder hexString = new StringBuilder();

for (byte b : hashBytes) {

String hex = Integer.toHexString(0xff & b);

if (hex.length() == 1) {

hexString.append('0');

}

hexString.append(hex);

}

return hexString.toString();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

return null;

}

}

}

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(PORT);

System.out.println("Server started. Waiting for clients...");

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress().getHostAddress());

ClientHandler clientHandler = new ClientHandler(clientSocket);

clientHandler.start();

}

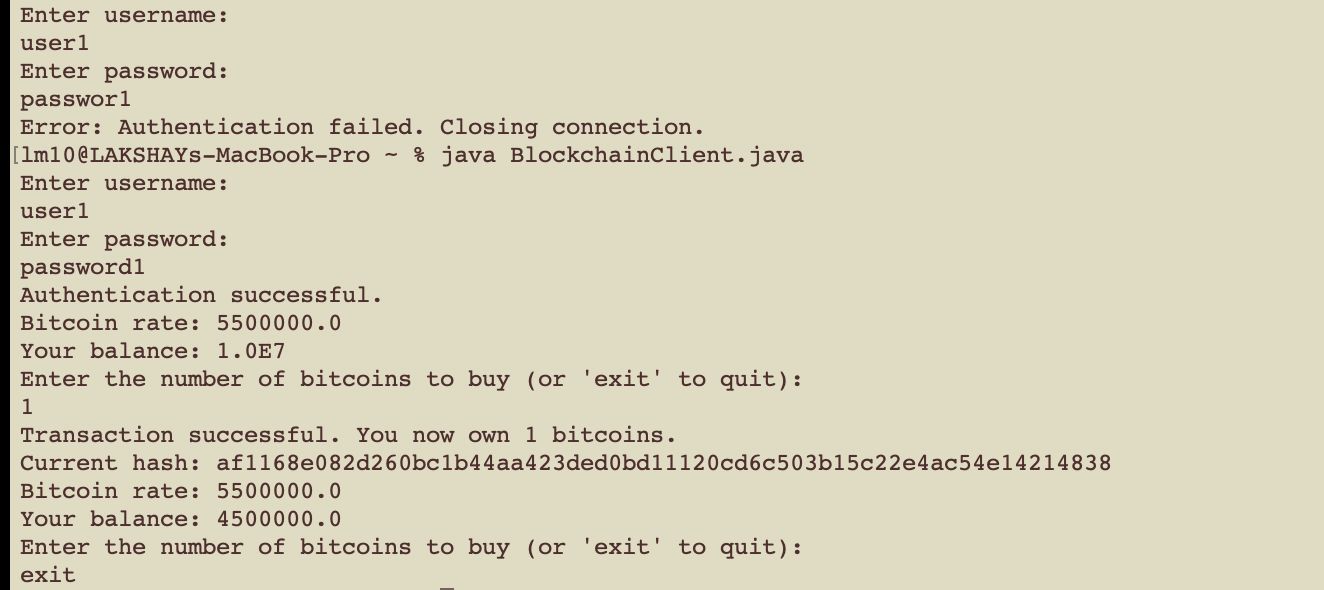
} catch (IOException e) {

e.printStackTrace();

}

}

}

****

**Program 3**

import javax.crypto.Cipher;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.DESKeySpec;

import javax.crypto.spec.SecretKeySpec;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.util.ArrayList;

import java.util.Base64;

import java.util.List;

public class MultipleBlockchains {

// Block class

static class Block {

private String data;

private String hash;

private String previousHash;

// Constructor

public Block(String data, String previousHash) {

this.data = data;

this.previousHash = previousHash;

this.hash = calculateHash();

}

// Calculate hash function

public String calculateHash() {

String dataToHash = data + previousHash;

return applySHA256(dataToHash);

}

// Apply SHA-256 hash function

private String applySHA256(String input) {

try {

MessageDigest digest = MessageDigest.getInstance("SHA-256");

byte[] hash = digest.digest(input.getBytes(StandardCharsets.UTF\_8));

StringBuilder hexString = new StringBuilder();

for (byte b : hash) {

String hex = Integer.toHexString(0xff & b);

if (hex.length() == 1) hexString.append('0');

hexString.append(hex);

}

return hexString.toString();

} catch (Exception e) {

throw new RuntimeException(e);

}

}

// Getters

public String getData() {

return data;

}

public String getHash() {

return hash;

}

public String getPreviousHash() {

return previousHash;

}

}

// Blockchain class

static class Blockchain {

private List<Block> chain;

private String previousHash;

// Constructor

public Blockchain() {

chain = new ArrayList<>();

previousHash = "0"; // Genesis block

}

// Add block to the chain

public void addBlock(String data) {

Block newBlock = new Block(data, previousHash);

chain.add(newBlock);

previousHash = newBlock.getHash();

}

// Get block by index

public Block getBlock(int index) {

return chain.get(index);

}

// Get size of the chain

public int size() {

return chain.size();

}

}

// Caesar Cipher encryption function

public static String caesarCipherEncrypt(String plainText, int shift) {

StringBuilder encryptedText = new StringBuilder();

for (char c : plainText.toCharArray()) {

if (Character.isLetter(c)) {

char base = Character.isLowerCase(c) ? 'a' : 'A';

encryptedText.append((char) ((c - base + shift) % 26 + base));

} else {

encryptedText.append(c);

}

}

return encryptedText.toString();

}

// DES encryption function

public static String desEncrypt(String plainText, byte[] key) throws Exception {

DESKeySpec desKeySpec = new DESKeySpec(key);

SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");

SecretKey secretKey = keyFactory.generateSecret(desKeySpec);

Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, secretKey);

byte[] encryptedBytes = cipher.doFinal(plainText.getBytes(StandardCharsets.UTF\_8));

return Base64.getEncoder().encodeToString(encryptedBytes);

}

// AES encryption function

public static String aesEncrypt(String plainText, byte[] key) throws Exception {

SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, secretKeySpec);

byte[] encryptedBytes = cipher.doFinal(plainText.getBytes(StandardCharsets.UTF\_8));

return Base64.getEncoder().encodeToString(encryptedBytes);

}

public static void main(String[] args) throws Exception {

// Test data

String[] data = {"Block 1 data", "Block 2 data", "Block 3 data"};

String secureKey = "your\_secure\_key";

byte[] desKey = MessageDigest.getInstance("SHA-256").digest(secureKey.getBytes(StandardCharsets.UTF\_8));

byte[] aesKey = MessageDigest.getInstance("SHA-256").digest(secureKey.getBytes(StandardCharsets.UTF\_8));

// Create and populate blockchains

Blockchain caesarBlockchain = new Blockchain();

Blockchain desBlockchain = new Blockchain();

Blockchain aesBlockchain = new Blockchain();

for (String blockData : data) {

// Encrypt data for each blockchain

String caesarEncryptedData = caesarCipherEncrypt(blockData, 3);

String desEncryptedData = desEncrypt(blockData, desKey);

String aesEncryptedData = aesEncrypt(blockData, aesKey);

// Add blocks to blockchains

caesarBlockchain.addBlock(caesarEncryptedData);

desBlockchain.addBlock(desEncryptedData);

aesBlockchain.addBlock(aesEncryptedData);

}

// Print blockchain data

System.out.println("Caesar Cipher Blockchain:");

printBlockchain(caesarBlockchain);

System.out.println("\nDES Blockchain:");

printBlockchain(desBlockchain);

System.out.println("\nAES Blockchain:");

printBlockchain(aesBlockchain);

}

// Helper method to print blockchain data

public static void printBlockchain(Blockchain blockchain) {

for (int i = 0; i < blockchain.size(); i++) {

Block block = blockchain.getBlock(i);

System.out.println("Block " + (i + 1) + ":");

System.out.println("Data: " + block.getData());

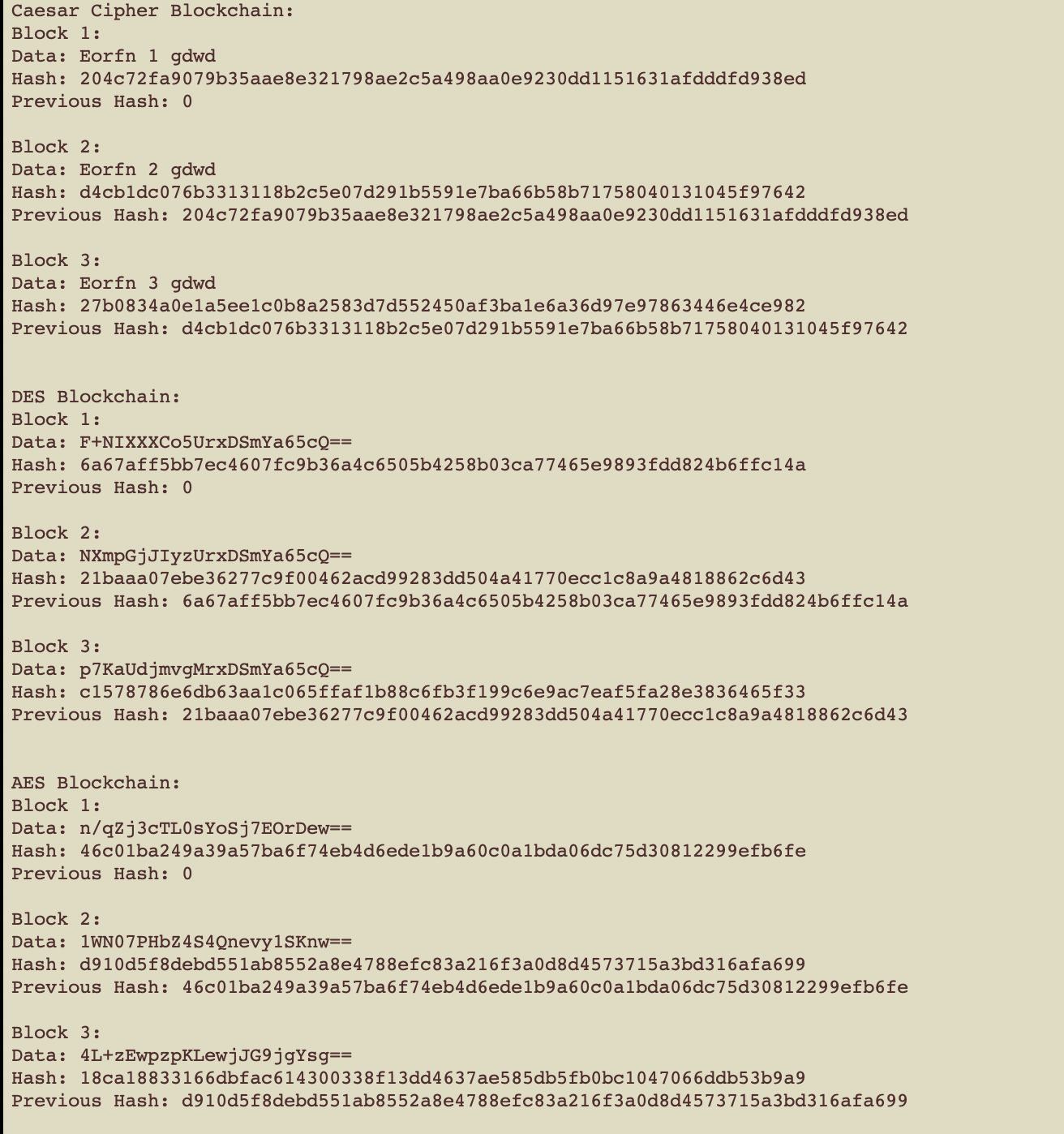
System.out.println("Hash: " + block.getHash());

System.out.println("Previous Hash: " + block.getPreviousHash());

System.out.println();

}

}

****}

**Program 4**

import java.security.\*;

public class TransactionAuthentication {

// Generate key pair

public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {

KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");

keyGen.initialize(2048); // key size

return keyGen.generateKeyPair();

}

// Sign transaction

public static String signTransaction(String transactionData, PrivateKey privateKey) throws Exception {

Signature sign = Signature.getInstance("SHA256withRSA");

sign.initSign(privateKey);

sign.update(transactionData.getBytes());

byte[] signatureBytes = sign.sign();

return bytesToHex(signatureBytes);

}

// Verify transaction signature

public static boolean verifyTransaction(String transactionData, String signature, PublicKey publicKey) throws Exception {

Signature verify = Signature.getInstance("SHA256withRSA");

verify.initVerify(publicKey);

verify.update(transactionData.getBytes());

byte[] signatureBytes = hexToBytes(signature);

return verify.verify(signatureBytes);

}

// Convert byte array to hex string

public static String bytesToHex(byte[] bytes) {

StringBuilder hexString = new StringBuilder();

for (byte b : bytes) {

String hex = Integer.toHexString(0xff & b);

if (hex.length() == 1) hexString.append('0');

hexString.append(hex);

}

return hexString.toString();

}

// Convert hex string to byte array

public static byte[] hexToBytes(String hexString) {

int len = hexString.length();

byte[] data = new byte[len / 2];

for (int i = 0; i < len; i += 2) {

data[i / 2] = (byte) ((Character.digit(hexString.charAt(i), 16) << 4)

+ Character.digit(hexString.charAt(i + 1), 16));

}

return data;

}

public static void main(String[] args) throws Exception {

// Generate key pair

KeyPair keyPair = generateKeyPair();

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();

// Transaction data

String transactionData = "Sample transaction data";

// Sign transaction

String signature = signTransaction(transactionData, privateKey);

// Verify transaction signature

boolean isVerified = verifyTransaction(transactionData, signature, publicKey);

// Output

System.out.println("Original Transaction Data: " + transactionData);

System.out.println("Generated Signature: " + signature);

System.out.println("Transaction Verification Result: " + (isVerified ? "Success" : "Failure"));

}

}



**Program 5**

Generate RSA Key Pairs:

$ openssl genpkey -algorithm RSA -out privatekey.pem -pkeyopt rsa\_keyge\_bits 1024

$ openssl rsa -pubout -in privatekey.pem -out publickey.pem

Encryption and Decryption using RSA:

$ openssl rsautl -encrypt -inkey publickey.pem -pubin -in message.txt -out message.rsa

$ openssl rsautl -decrypt -inkey privatekey.pem -in message.rsa -out message.dec

Generate ECC Key Pairs:

$ openssl ecparam -genkey -name secp256k1 -noout -out ecc\_private\_key.pem

$ openssl ec -in ecc\_private\_key.pem -pubout -out ecc\_public\_key.pem

Encryption and Decryption using ECC:

$ openssl pkeyutl -encrypt -inkey publickey.pem -pubin -in message.txt -out message.ecc

$ openssl pkeyutl -decrypt -inkey privatekey.pem -in message.ecc -out message.txt

**Program 6**

import java.security.\*;

import java.security.spec.ECGenParameterSpec;

import java.util.Base64;

import java.util.Scanner;

public class BlockchainWallet {

private PrivateKey privateKey;

private PublicKey publicKey;

private float balance;

public BlockchainWallet() {

generateKeyPair();

balance = 100.0f; // Initially adding 100 to each account

}

private void generateKeyPair() {

try {

KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC");

SecureRandom random = SecureRandom.getInstanceStrong();

ECGenParameterSpec ecSpec = new ECGenParameterSpec("secp256k1");

keyGen.initialize(ecSpec, random);

KeyPair keyPair = keyGen.generateKeyPair();

privateKey = keyPair.getPrivate();

publicKey = keyPair.getPublic();

} catch (Exception e) {

e.printStackTrace();

}

}

public String getPublicKey() {

return Base64.getEncoder().encodeToString(publicKey.getEncoded());

}

public void sendMoney(BlockchainWallet recipient, float amount) {

if (balance >= amount) {

balance -= amount;

recipient.receiveMoney(amount);

System.out.println("Transaction Successful! Sent " + amount + " from " + getPublicKey() + " to " + recipient.getPublicKey());

} else {

System.out.println("Insufficient funds to send.");

}

}

public void receiveMoney(float amount) {

balance += amount;

}

public float getBalance() {

return balance;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BlockchainWallet walletA = new BlockchainWallet();

BlockchainWallet walletB = new BlockchainWallet();

boolean running = true;

while (running) {

System.out.println("Choose an option:");

System.out.println("1. Check Wallet Balance");

System.out.println("2. Send Money");

System.out.println("3. Exit");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.println("WalletA Balance: " + walletA.getBalance());

System.out.println("WalletB Balance: " + walletB.getBalance());

break;

case 2:

System.out.println("Enter the amount to send from WalletA to WalletB:");

float amountToSend = scanner.nextFloat();

walletA.sendMoney(walletB, amountToSend);

break;

case 3:

running = false;

break;

default:

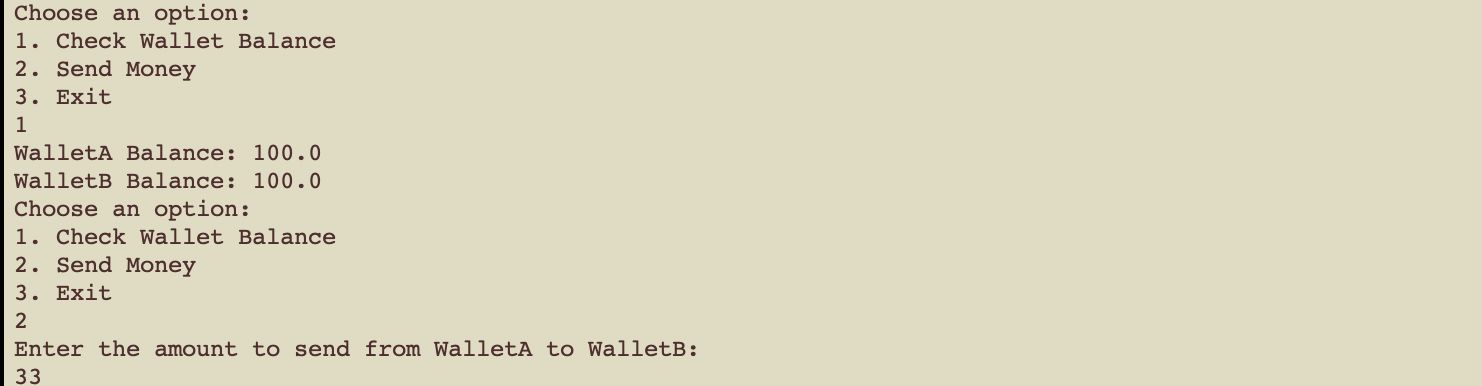
System.out.println("Invalid choice. Please choose a valid option.");

}

}

scanner.close();

}

****}

**Program 7**

$geth --mine --minerthreads 4

CPU Mining

$ etherminer -M -C

GPU Mining

$ etherminer -G

