**PROJECT 3**

**Project3Task0 Execution**

/Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=57283:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task0/out/production/Project3Task0 blockchaintask0.Blockchain

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Experimented with 2,000,000 hashes.

Approximate hashes per second on this machine: 285714285

Expected total hashes required for the whole chain: 256.000000

Nonce for most recent block: 838

Chain hash: 00d363dfaa7f466ac5a2037733e1400307ea35395702dba095ad4684512d07ee

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Alice pays Bob 100 DSCoin

Total execution time to add this block was 245 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Bob pays Carol 20 DSCoin

Total execution time to add this block was 4 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Carol pays Donna 10 DSCoin

Total execution time to add this block was 59 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 06:06:38.331","Tx ": "Genesis","PrevHash" : "0","nonce" : 838,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 06:07:37.535","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00d363dfaa7f466ac5a2037733e1400307ea35395702dba095ad4684512d07ee","nonce" : 123295,"difficulty": 4},{"index" : 2,"time stamp " : "2024-10-31 06:07:56.666","Tx ": "Bob pays Carol 20 DSCoin","PrevHash" : "00001361191d2baa8548969e272855d2500714d9c21dde09ebfd07d7d10550d0","nonce" : 1259,"difficulty": 4},{"index" : 3,"time stamp " : "2024-10-31 06:08:11.669","Tx ": "Carol pays Donna 10 DSCoin","PrevHash" : "00007f4a2f4f1c71c56a6ddb26a68a98aa728eb2f6bfbb683ef00c790170f073","nonce" : 33783,"difficulty": 4}], "chainHash":"0000b49a907c9523b68cf13de78a422ad011ea045c4736de5a6dd933739cb6db"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: TRUE

Total execution time required to verify the chain was 1 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 4

Corrupt the Blockchain

Enter block ID of block to corrupt: 2

Enter new data for block 2: Bob pays Tony 30 DSCoin

Block 2 now holds Bob pays Tony 30 DSCoin

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 06:06:38.331","Tx ": "Genesis","PrevHash" : "0","nonce" : 838,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 06:07:37.535","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00d363dfaa7f466ac5a2037733e1400307ea35395702dba095ad4684512d07ee","nonce" : 123295,"difficulty": 4},{"index" : 2,"time stamp " : "2024-10-31 06:07:56.666","Tx ": "Bob pays Tony 30 DSCoin","PrevHash" : "00001361191d2baa8548969e272855d2500714d9c21dde09ebfd07d7d10550d0","nonce" : 1259,"difficulty": 4},{"index" : 3,"time stamp " : "2024-10-31 06:08:11.669","Tx ": "Carol pays Donna 10 DSCoin","PrevHash" : "00007f4a2f4f1c71c56a6ddb26a68a98aa728eb2f6bfbb683ef00c790170f073","nonce" : 33783,"difficulty": 4}], "chainHash":"0000b49a907c9523b68cf13de78a422ad011ea045c4736de5a6dd933739cb6db"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: FALSE

Improper hash on node 2. Does not begin with 0000

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 5

Repairing the entire chain

Total execution time required to repair the chain was 154 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: TRUE

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 06:06:38.331","Tx ": "Genesis","PrevHash" : "0","nonce" : 838,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 06:07:37.535","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00d363dfaa7f466ac5a2037733e1400307ea35395702dba095ad4684512d07ee","nonce" : 123295,"difficulty": 4},{"index" : 2,"time stamp " : "2024-10-31 06:07:56.666","Tx ": "Bob pays Tony 30 DSCoin","PrevHash" : "00001361191d2baa8548969e272855d2500714d9c21dde09ebfd07d7d10550d0","nonce" : 117624,"difficulty": 4},{"index" : 3,"time stamp " : "2024-10-31 06:08:11.669","Tx ": "Carol pays Donna 10 DSCoin","PrevHash" : "000099d7e379160985c155a2bbc068721fcd5153804d1600f2fcbe214ad85b58","nonce" : 77255,"difficulty": 4}], "chainHash":"000039c65f033e09e8c8d03324efbf10202bad0a919836c3c4b1bf9cfaf95533"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 6

Exiting...

Process finished with exit code 0

**Project3Task0 Block.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
  
package blockchaintask0;  
  
import java.math.BigInteger;  
import java.security.MessageDigest;  
import java.sql.Timestamp;  
  
*/\*\*  
 \* The Block class represents an individual block within the blockchain.  
 \* Each block has an index, timestamp, data, previous hash, nonce, and difficulty.  
 \*/*public class Block {  
 private int index; // Position of the block within the blockchain  
 private Timestamp timestamp; // Time when the block was generated  
 private String data; // Transaction data stored in the block  
 private String previousHash; // Hash of the preceding block in the chain  
 private BigInteger nonce; // Nonce value determined by proof of work  
 private int difficulty; // Mining difficulty level for the block  
  
 */\*\*  
 \* Constructor to initialize a new Block.  
 \*  
 \* @param index Position within the chain. Genesis block is at 0.  
 \* @param timestamp Creation time of the block.  
 \* @param data Transaction details stored in the block.  
 \* @param difficulty Required number of leading hex zeroes for the hash.  
 \*/* public Block(int index, Timestamp timestamp, String data, int difficulty) {  
 this.index = index;  
 this.timestamp = timestamp;  
 this.data = data;  
 this.difficulty = difficulty;  
 this.previousHash = "0";  
 this.nonce = BigInteger.*ZERO*;  
 }  
  
 */\*\*  
 \* Calculates the SHA-256 hash of the block based on its properties.  
 \*  
 \* @return a String representing the hash in hexadecimal format.  
 \*/* public String calculateHash() {  
 try {  
 String input = index + timestamp.toString() + data + previousHash + nonce + difficulty;  
 MessageDigest digest = MessageDigest.*getInstance*("SHA-256");  
 byte[] hashBytes = digest.digest(input.getBytes("UTF-8"));  
 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 (Exception e) {  
 throw new RuntimeException("Error calculating hash", e);  
 }  
 }  
  
 */\*\*  
 \* Finds a valid hash by incrementing the nonce until the hash meets the difficulty.  
 \*  
 \* @return the valid hash with the required number of leading hex zeroes.  
 \*/* public String mineBlock() {  
 String target = "0".repeat(difficulty);  
 while (!calculateHash().startsWith(target)) {  
 nonce = nonce.add(BigInteger.*ONE*);  
 }  
 return calculateHash();  
 }  
  
 */\*\*  
 \* Gets the nonce for this block.  
 \*  
 \* @return a BigInteger representing the nonce for this block.  
 \*/* public BigInteger getNonce() {  
 return nonce;  
 }  
  
 */\*\*  
 \* Gets the difficulty level of the block.  
 \*  
 \* @return the difficulty of this block.  
 \*/* public int getDifficulty() {  
 return difficulty;  
 }  
  
 */\*\*  
 \* Sets the difficulty level required for this block.  
 \*  
 \* @param difficulty the difficulty level specifying the required leading zeroes in the hash.  
 \*/* public void setDifficulty(int difficulty) {  
 this.difficulty = difficulty;  
 }  
  
 */\*\*  
 \* Gets the previous hash for this block.  
 \*  
 \* @return the hash of the preceding block in the chain.  
 \*/* public String getPreviousHash() {  
 return previousHash;  
 }  
  
 */\*\*  
 \* Sets the previous hash (hash pointer) of this block's parent.  
 \*  
 \* @param previousHash the hash of the preceding block.  
 \*/* public void setPreviousHash(String previousHash) {  
 this.previousHash = previousHash;  
 }  
  
 */\*\*  
 \* Gets the index of the block.  
 \*  
 \* @return the index of this block within the chain.  
 \*/* public int getIndex() {  
 return index;  
 }  
  
 */\*\*  
 \* Sets the index of the block.  
 \*  
 \* @param index the position of this block within the chain.  
 \*/* public void setIndex(int index) {  
 this.index = index;  
 }  
  
 */\*\*  
 \* Gets the timestamp of this block.  
 \*  
 \* @return the timestamp representing when this block was created.  
 \*/* public Timestamp getTimestamp() {  
 return timestamp;  
 }  
  
 */\*\*  
 \* Sets the timestamp of the block.  
 \*  
 \* @param timestamp the creation time of this block.  
 \*/* public void setTimestamp(Timestamp timestamp) {  
 this.timestamp = timestamp;  
 }  
  
 */\*\*  
 \* Gets the transaction data stored in this block.  
 \*  
 \* @return the transaction details of this block.  
 \*/* public String getData() {  
 return data;  
 }  
  
 */\*\*  
 \* Sets the transaction data for this block.  
 \*  
 \* @param data the transaction details to be included in the block.  
 \*/* public void setData(String data) {  
 this.data = data;  
 }  
  
 */\*\*  
 \* Provides a JSON-like string representation of the block's data.  
 \*  
 \* @return a JSON-like string with the block's data fields.  
 \*/* @Override  
 public String toString() {  
 return "{\"index\" : " + index + "," +  
 "\"time stamp \" : \"" + timestamp + "\"," +  
 "\"Tx \": \"" + data + "\"," +  
 "\"PrevHash\" : \"" + previousHash + "\"," +  
 "\"nonce\" : " + nonce + "," +  
 "\"difficulty\": " + difficulty + "}";  
 }  
}

**Task0 Blockchain.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
  
package blockchaintask0;  
  
import java.sql.Timestamp;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
*/\*\*  
 \* The Blockchain class represents a simple blockchain structure.  
 \* It contains methods to add blocks, validate the chain, corrupt blocks, and repair the chain.  
 \*/*public class Blockchain {  
 private ArrayList<Block> blockchain; // List to hold blocks in the blockchain  
 private String chainHash; // Hash of the most recently added block  
 private int hashesPerSecond; // Approximate hash rate in hashes per second  
  
 */\*\*  
 \* Initializes the blockchain with a genesis block and sets the initial chain hash.  
 \*/* public Blockchain() {  
 blockchain = new ArrayList<>();  
 chainHash = "";  
  
 // Create and mine the genesis block (initial block)  
 Block genesisBlock = new Block(0, new Timestamp(System.*currentTimeMillis*()), "Genesis", 2);  
 genesisBlock.mineBlock();  
 chainHash = genesisBlock.calculateHash();  
 blockchain.add(genesisBlock);  
  
 // Calculate the approximate hashes per second on this machine  
 computeHashesPerSecond();  
 }  
  
 */\*\*  
 \* Adds a new block to the blockchain.  
 \*  
 \* @param transactionData the transaction data to be stored in the block  
 \* @param difficultyLevel the mining difficulty level for the block  
 \*/* public void addBlock(String transactionData, int difficultyLevel) {  
 long startTime = System.*currentTimeMillis*();  
 Block newBlock = new Block(blockchain.size(), new Timestamp(System.*currentTimeMillis*()), transactionData, difficultyLevel);  
 newBlock.setPreviousHash(chainHash);  
 newBlock.mineBlock();  
 chainHash = newBlock.calculateHash();  
 blockchain.add(newBlock);  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.printf("Total execution time to add this block was %d milliseconds%n", (endTime - startTime));  
 }  
  
 */\*\*  
 \* Checks the validity of the entire blockchain.  
 \*  
 \* @return "TRUE" if the blockchain is valid, otherwise an error message  
 \*/* public void isChainValid() {  
 long startTime = System.*currentTimeMillis*();  
 String previousHash = "0";  
  
 for (int i = 0; i < blockchain.size(); i++) {  
 Block currentBlock = blockchain.get(i);  
 String calculatedHash = currentBlock.calculateHash();  
  
 // Check if the previous hash matches the stored previous hash  
 if (!currentBlock.getPreviousHash().equals(previousHash)) {  
 System.*out*.println("Chain verification: FALSE");  
 System.*out*.printf("Improper hash on node %d. Does not match previous hash.%n", i);  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.printf("Total execution time required to verify the chain was %d milliseconds%n", (endTime - startTime));  
 return;  
 }  
  
 // Check if the current hash meets the difficulty requirement  
 if (!calculatedHash.startsWith("0".repeat(currentBlock.getDifficulty()))) {  
 System.*out*.println("Chain verification: FALSE");  
 System.*out*.printf("Improper hash on node %d. Does not begin with %s%n", i, "0".repeat(currentBlock.getDifficulty()));  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.printf("Total execution time required to verify the chain was %d milliseconds%n", (endTime - startTime));  
 return;  
 }  
  
 // Update previousHash for the next block in the chain  
 previousHash = calculatedHash;  
 }  
  
 // If all checks pass, print TRUE  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.println("Chain verification: TRUE");  
 System.*out*.printf("Total execution time required to verify the chain was %d milliseconds%n", (endTime - startTime));  
 }  
  
  
 */\*\*  
 \* Alters the data in a specific block to simulate corruption.  
 \*  
 \* @param index the index of the block to be corrupted  
 \* @param newTransactionData the new transaction data for the block  
 \*/* public void corruptBlock(int index, String newTransactionData) {  
 if (index >= 0 && index < blockchain.size()) {  
 blockchain.get(index).setData(newTransactionData);  
 System.*out*.printf("Block %d now holds %s%n", index, newTransactionData);  
 } else {  
 System.*out*.println("Invalid block index.");  
 }  
 }  
  
 */\*\*  
 \* Repairs the blockchain by recalculating proof of work for each block in sequence.  
 \*/* public void repairChain() {  
 long startTime = System.*currentTimeMillis*();  
 for (int i = 1; i < blockchain.size(); i++) {  
 Block currentBlock = blockchain.get(i);  
 currentBlock.setPreviousHash(blockchain.get(i - 1).calculateHash());  
 currentBlock.mineBlock();  
 }  
 chainHash = blockchain.get(blockchain.size() - 1).calculateHash();  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.printf("Total execution time required to repair the chain was %d milliseconds%n", (endTime - startTime));  
 }  
  
 */\*\*  
 \* Estimates the hashes per second by timing a set number of hash calculations.  
 \*/* private void computeHashesPerSecond() {  
 long startTime = System.*currentTimeMillis*();  
 int numHashes = 2000000;  
 for (int i = 0; i < numHashes; i++) {  
 "test".hashCode();  
 }  
 long endTime = System.*currentTimeMillis*();  
 long duration = endTime - startTime;  
 hashesPerSecond = (int) (numHashes / (duration / 1000.0));  
 }  
  
 */\*\*  
 \* Displays the basic status of the blockchain.  
 \*/* public void displayStatus() {  
 System.*out*.printf("Current size of chain: %d%n", blockchain.size());  
 System.*out*.printf("Difficulty of most recent block: %d%n", blockchain.get(blockchain.size() - 1).getDifficulty());  
 System.*out*.printf("Total difficulty for all blocks: %d%n", blockchain.stream().mapToInt(Block::getDifficulty).sum());  
 System.*out*.println("Experimented with 2,000,000 hashes.");  
 System.*out*.printf("Approximate hashes per second on this machine: %d%n", hashesPerSecond);  
 System.*out*.printf("Expected total hashes required for the whole chain: %.6f%n", blockchain.stream().mapToDouble(b -> Math.*pow*(16, b.getDifficulty())).sum());  
 System.*out*.printf("Nonce for most recent block: %d%n", blockchain.get(blockchain.size() - 1).getNonce());  
 System.*out*.printf("Chain hash: %s%n", chainHash);  
 }  
  
 */\*\*  
 \* Provides a JSON-like string representation of the entire blockchain.  
 \*  
 \* @return a JSON string showing the blockchain contents  
 \*/* @Override  
 public String toString() {  
 StringBuilder builder = new StringBuilder();  
 builder.append("{\"ds\_chain\" : [");  
 for (int i = 0; i < blockchain.size(); i++) {  
 builder.append(blockchain.get(i).toString());  
 if (i < blockchain.size() - 1) {  
 builder.append(",");  
 }  
 }  
 builder.append("], \"chainHash\":\"").append(chainHash).append("\"}");  
 return builder.toString();  
 }  
  
 */\*\*  
 \* Main interactive console for interacting with the blockchain.  
 \*  
 \* @param args command line arguments (unused)  
 \*/* public static void main(String[] args) {  
 Blockchain blockchain = new Blockchain();  
 Scanner scanner = new Scanner(System.*in*);  
 int choice;  
  
 /\*  
 As the difficulty level increases, the time taken for certain blockchain operations increases substantially, particularly for addBlock() and repairChain() \*/  
   
 do {  
 System.*out*.println("\n0. View basic blockchain status.");  
 System.*out*.println("1. Add a transaction to the blockchain.");  
 System.*out*.println("2. Verify the blockchain.");  
 System.*out*.println("3. View the blockchain.");  
 System.*out*.println("4. Corrupt the chain.");  
 System.*out*.println("5. Hide the corruption by repairing the chain.");  
 System.*out*.println("6. Exit");  
 System.*out*.print("Enter your choice: ");  
 choice = scanner.nextInt();  
  
 switch (choice) {  
 case 0:  
 blockchain.displayStatus();  
 break;  
 case 1:  
 System.*out*.print("Enter difficulty > 1: ");  
 int difficulty = scanner.nextInt();  
 scanner.nextLine();  
 System.*out*.print("Enter transaction: ");  
 String transactionData = scanner.nextLine();  
 blockchain.addBlock(transactionData, difficulty);  
 break;  
 case 2:  
 System.*out*.println("Verifying entire chain");  
 blockchain.isChainValid();  
 break;  
 case 3:  
 System.*out*.println("View the Blockchain");  
 System.*out*.println(blockchain);  
 break;  
 case 4:  
 System.*out*.println("Corrupt the Blockchain");  
 System.*out*.print("Enter block ID of block to corrupt: ");  
 int blockId = scanner.nextInt();  
 scanner.nextLine();  
 System.*out*.print("Enter new data for block " + blockId + ": ");  
 String newData = scanner.nextLine();  
 blockchain.corruptBlock(blockId, newData);  
 break;  
 case 5:  
 System.*out*.println("Repairing the entire chain");  
 blockchain.repairChain();  
 break;  
 case 6:  
 System.*out*.println("Exiting...");  
 break;  
 default:  
 System.*out*.println("Invalid option. Please try again.");  
 break;  
 }  
 } while (choice != 6);  
  
 scanner.close();  
 }  
}  
  
//Referenced ChatGPT for documentation

**Experiment by adding new blocks with increasing difficulties**

**A computer screen shot of a program

Description automatically generated**

**Output:**/Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=58506:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task0/out/production/Project3Task0 blockchaintask0.Blockchain

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Experimented with 2,000,000 hashes.

Approximate hashes per second on this machine: 500000000

Expected total hashes required for the whole chain: 256.000000

Nonce for most recent block: 617

Chain hash: 00ccd7e9f700dc75246bce0e672bc47173be6e189143bc0bd49ba1db7003206e

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 2

Enter transaction: Alice pays Bob 100 DSCoin

Total execution time to add this block was 14 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Bob pays Carol 20 DSCoin

Total execution time to add this block was 196 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 07:28:42.034","Tx ": "Genesis","PrevHash" : "0","nonce" : 617,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 07:29:29.885","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00ccd7e9f700dc75246bce0e672bc47173be6e189143bc0bd49ba1db7003206e","nonce" : 447,"difficulty": 2},{"index" : 2,"time stamp " : "2024-10-31 07:29:57.735","Tx ": "Bob pays Carol 20 DSCoin","PrevHash" : "000bebaa93b64bd1f65727ba2a53eef942acc793a5ec7c00870fe966af304e97","nonce" : 91677,"difficulty": 4}], "chainHash":"000033abce7beedae35cba8541fdb805fb0e918e5b42e5316f76e601162a83a9"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: TRUE

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 6

Exiting...

Process finished with exit code 0

**Task 1 Client Source Code in the file ClientTCP.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
package org.example;  
  
import com.google.gson.Gson;  
  
import java.io.\*;  
import java.net.Socket;  
  
*/\*\*  
 \* ClientTCP serves as a client to interact with the blockchain server over TCP sockets.  
 \* It sends JSON-formatted requests for blockchain operations, such as adding a block,  
 \* verifying the chain, viewing the blockchain, and repairing the chain.  
 \*/*public class ClientTCP {  
 public static void main(String args[]) {  
 Socket clientSocket = null;  
 try {  
 // Connect to the server socket running on localhost:7777  
 int serverPort = 7777;  
 clientSocket = new Socket("localhost", serverPort);  
  
 // Set up input and output streams for communication with the server  
 BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
 PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));  
  
 // Set up Gson for JSON serialization/deserialization  
 Gson gson = new Gson();  
 BufferedReader typed = new BufferedReader(new InputStreamReader(System.*in*));  
 int choice;  
 int blockId = 0; // Declare blockId for corrupting specific blocks  
  
 do {  
 // Display menu options  
 System.*out*.println("\n0. View basic blockchain status.");  
 System.*out*.println("1. Add a transaction to the blockchain.");  
 System.*out*.println("2. Verify the blockchain.");  
 System.*out*.println("3. View the blockchain.");  
 System.*out*.println("4. Corrupt the chain.");  
 System.*out*.println("5. Hide the corruption by repairing the chain.");  
 System.*out*.println("6. Exit");  
 System.*out*.print("Enter your choice: ");  
 String choiceInput = typed.readLine().trim();  
 choice = Integer.*parseInt*(choiceInput);  
  
 RequestMessage requestMessage = null;  
 switch (choice) {  
 case 0: // Retrieve blockchain status  
 requestMessage = new RequestMessage("getStatus", null, 0);  
 break;  
 case 1: // Add a new block to the blockchain  
 System.*out*.print("Enter difficulty > 1: ");  
 int difficulty = Integer.*parseInt*(typed.readLine());  
 System.*out*.print("Enter transaction: ");  
 String transaction = typed.readLine();  
 requestMessage = new RequestMessage("addBlock", transaction, difficulty);  
 break;  
 case 2: // Verify the integrity of the blockchain  
 requestMessage = new RequestMessage("verifyChain", null, 0);  
 break;  
 case 3: // View the entire blockchain  
 requestMessage = new RequestMessage("viewChain", null, 0);  
 break;  
 case 4: // Corrupt a block in the blockchain  
 System.*out*.println("Corrupt the Blockchain");  
 System.*out*.print("Enter block ID of block to corrupt: ");  
 blockId = Integer.*parseInt*(typed.readLine()); // Initialize blockId here  
 System.*out*.print("Enter new data for block " + blockId + ": ");  
 String newData = typed.readLine();  
 requestMessage = new RequestMessage("corruptBlock", newData, blockId);  
 break;  
 case 5: // Repair the blockchain  
 requestMessage = new RequestMessage("repairChain", null, 0);  
 break;  
 case 6: // Exit  
 System.*out*.println("Exiting...");  
 break;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 break;  
 }  
  
 // Send the request to the server and process the response  
 if (requestMessage != null) {  
 String jsonRequest = gson.toJson(requestMessage);  
 out.println(jsonRequest);  
 out.flush();  
 String jsonResponse = in.readLine();  
 ResponseMessage responseMessage = gson.fromJson(jsonResponse, ResponseMessage.class);  
  
 switch (choice) {  
 case 0: // Display blockchain status  
 System.*out*.println("Current size of chain: " + responseMessage.getChainSize());  
 System.*out*.println("Difficulty of most recent block: " + responseMessage.getDifficulty());  
 System.*out*.println("Total difficulty for all blocks: " + responseMessage.getTotalDifficulty());  
 System.*out*.println("Experimented with 2,000,000 hashes.");  
 System.*out*.println("Approximate hashes per second on this machine: " + responseMessage.getHashesPerSecond());  
 System.*out*.println("Expected total hashes required for the whole chain: " + responseMessage.getTotalExpectedHashes());  
 System.*out*.println("Nonce for most recent block: " + responseMessage.getNonce());  
 System.*out*.println("Chain hash: " + responseMessage.getChainHash());  
 break;  
 case 1: // Display block creation time  
 long blockCreationTime = responseMessage.getEndTime() - responseMessage.getStartTime();  
 System.*out*.println("Total execution time to add this block was " + blockCreationTime + " milliseconds");  
 break;  
 case 2: // Display chain verification result  
 long verificationTime = responseMessage.getEndTime() - responseMessage.getStartTime();  
 System.*out*.println("Verifying entire chain");  
 System.*out*.println("Chain verification: " + responseMessage.getMessage());  
 System.*out*.println("Total execution time required to verify the chain was " + verificationTime + " milliseconds");  
 break;  
 case 3: // Display the blockchain  
 System.*out*.println("View the Blockchain");  
 System.*out*.println(responseMessage.getChainHash());  
 break;  
 case 4: // Display confirmation of block corruption  
 System.*out*.println("Block " + blockId + " now holds " + requestMessage.getTransactionData());  
 break;  
 case 5: // Display chain repair time  
 long repairTime = responseMessage.getEndTime() - responseMessage.getStartTime();  
 System.*out*.println("Repairing the entire chain");  
 System.*out*.println("Total execution time required to repair the chain was " + repairTime + " milliseconds");  
 break;  
 case 6: // Exit  
 break;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 } while (choice != 6);  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
 } finally {  
 try {  
 if (clientSocket != null) {  
 clientSocket.close();  
 }  
 } catch (IOException e) {  
 // Ignore exception on close  
 }  
 }  
 }  
}

**Task 1 Server Source Code in the file ServerTCP.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
package org.example;  
  
import com.google.gson.Gson;  
import java.io.BufferedWriter;  
import java.io.IOException;  
import java.io.OutputStreamWriter;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.sql.Timestamp;  
import java.util.Scanner;  
  
*/\*\*  
 \* ServerTCP handles client requests over TCP sockets, manages the blockchain on the server,  
 \* and processes various blockchain operations.  
 \*/*public class ServerTCP {  
 public static void main(String[] args) {  
 try {  
 // Set up server socket to listen on port 7777  
 int port = 7777;  
 ServerSocket serverSocket = new ServerSocket(port);  
 System.*out*.println("Blockchain Server Running");  
  
 // Initialize blockchain with a genesis block  
 Blockchain blockchain = new Blockchain();  
  
 // Continuously listen for incoming client connections  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 try (Scanner inputScanner = new Scanner(clientSocket.getInputStream());  
 PrintWriter outputWriter = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())))) {  
  
 Gson gson = new Gson();  
  
 while (inputScanner.hasNextLine()) {  
 // Process each incoming client request  
 String requestJson = inputScanner.nextLine();  
 RequestMessage request = gson.fromJson(requestJson, RequestMessage.class);  
 String responseJson = *processRequest*(request, blockchain, gson);  
 outputWriter.println(responseJson);  
 outputWriter.flush();  
  
 // Log request and response details for debugging  
 System.*out*.println("\nWe have a visitor");  
 System.*out*.println("THE JSON REQUEST MESSAGE IS: " + requestJson);  
 System.*out*.println("THE JSON RESPONSE MESSAGE IS: " + responseJson);  
 System.*out*.println("Number of Blocks on Chain: " + blockchain.getSize());  
 }  
 } catch (IOException e) {  
 System.*out*.println("Error handling client: " + e.getMessage());  
 } finally {  
 System.*out*.println("Client disconnected");  
 clientSocket.close();  
 }  
 }  
 } catch (IOException e) {  
 System.*out*.println("Server error: " + e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* Processes incoming requests and generates appropriate responses.  
 \*  
 \* @param request The request message received from the client.  
 \* @param blockchain The blockchain instance to perform operations on.  
 \* @param gson Gson instance for JSON processing.  
 \* @return JSON-formatted response message.  
 \*/* private static String processRequest(RequestMessage request, Blockchain blockchain, Gson gson) {  
 long startTime, endTime;  
 String responseJson;  
  
 switch (request.getAction()) {  
 case "getStatus": // Retrieve blockchain status  
 startTime = System.*currentTimeMillis*();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Viewing status", blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "addBlock": // Add a new block to the blockchain  
 startTime = System.*currentTimeMillis*();  
 Block newBlock = new Block(blockchain.getSize(), new Timestamp(System.*currentTimeMillis*()), request.getTransactionData(), request.getDifficultyOrIndex());  
 blockchain.addBlock(request.getTransactionData(), request.getDifficultyOrIndex());  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Block added", blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "verifyChain": // Verify the blockchain's integrity  
 startTime = System.*currentTimeMillis*();  
 String validationResult = blockchain.isChainValid();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", validationResult, blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "viewChain": // View the entire blockchain  
 startTime = System.*currentTimeMillis*();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Viewing blockchain", blockchain.toString(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "corruptBlock": // Corrupt a specific block in the blockchain  
 startTime = System.*currentTimeMillis*();  
 int blockIndex = request.getDifficultyOrIndex();  
 if (blockIndex >= 0 && blockIndex < blockchain.getSize()) {  
 blockchain.corruptBlock(blockIndex, request.getTransactionData());  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Block " + blockIndex + " corrupted with new data",  
 blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 } else {  
 responseJson = gson.toJson(new ResponseMessage(  
 "error", "Invalid block index", null, blockchain.getSize(),  
 0, blockchain.getTotalDifficulty(), 0, 0, null, 0L, 0L));  
 }  
 break;  
  
 case "repairChain": // Repair the blockchain by recalculating hashes  
 startTime = System.*currentTimeMillis*();  
 blockchain.repairChain();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Blockchain repaired", blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 default: // Invalid action  
 responseJson = gson.toJson(new ResponseMessage(  
 "error", "Invalid action", null, -1, 0, 0, 0, 0, null, 0L, 0L));  
 break;  
 }  
 return responseJson;  
 }  
}

//Referenced ChatGPT for error solving

**RequestMessage.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
package org.example;  
  
*/\*\*  
 \* Represents a request message sent from the client to the server,  
 \* specifying the action, data (transaction information), and difficulty level.  
 \*/*public class RequestMessage {  
 private String action; // The action to be performed, e.g., "addBlock", "isChainValid", "corruptBlock"  
 private String transactionData; // Data related to the transaction, if applicable  
 private int difficultyOrIndex; // Difficulty level for mining or block index for specific actions  
  
 */\*\*  
 \* Constructs a RequestMessage with specified action, transaction data, and difficulty level.  
 \*  
 \* @param action the action to be performed by the server  
 \* @param transactionData the transaction data or content relevant to the action  
 \* @param difficultyOrIndex the difficulty level or index for the block depending on the action  
 \*/* public RequestMessage(String action, String transactionData, int difficultyOrIndex) {  
 this.action = action;  
 this.transactionData = transactionData;  
 this.difficultyOrIndex = difficultyOrIndex;  
 }  
  
 */\*\*  
 \* Gets the action associated with the request.  
 \*  
 \* @return the action as a string  
 \*/* public String getAction() {  
 return action;  
 }  
  
 */\*\*  
 \* Gets the transaction data associated with the request.  
 \*  
 \* @return the transaction data as a string  
 \*/* public String getTransactionData() {  
 return transactionData;  
 }  
  
 */\*\*  
 \* Gets the difficulty level or block index associated with the request.  
 \*  
 \* @return an integer representing either the difficulty or block index  
 \*/* public int getDifficultyOrIndex() {  
 return difficultyOrIndex;  
 }  
  
 */\*\*  
 \* Returns a string representation of the request message in JSON format.  
 \*  
 \* @return a JSON-like string with the action, transaction data, and difficulty or index  
 \*/* @Override  
 public String toString() {  
 return "{\"action\":\"" + action + "\", \"transactionData\":\"" + transactionData + "\", \"difficultyOrIndex\":" + difficultyOrIndex + "}";  
 }  
}

**ResponseMessage.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
package org.example;  
  
import java.math.BigInteger;  
  
*/\*\*  
 \* Represents a response message sent from the server to the client.  
 \* Contains details about the blockchain's status or the result of an operation.  
 \*/*public class ResponseMessage {  
 private String status; // Status of the response (e.g., "success" or "error")  
 private String message; // Message detailing the response  
 private String chainHash; // Current hash of the blockchain  
 private int chainSize; // Total number of blocks in the blockchain  
 private int difficulty; // Difficulty level of the last block  
 private int totalDifficulty; // Aggregate difficulty of all blocks in the blockchain  
 private int hashesPerSecond; // Estimated hashing speed of the system  
 private double totalExpectedHashes; // Total expected hashes for the blockchain based on difficulty  
 private BigInteger nonce; // Nonce of the last block in the blockchain  
 private long startTime; // Timestamp for the start of an operation  
 private long endTime; // Timestamp for the end of an operation  
  
 */\*\*  
 \* Constructs a new ResponseMessage with the specified details.  
 \*  
 \* @param status The status of the response  
 \* @param message A message describing the response  
 \* @param chainHash The hash of the blockchain  
 \* @param chainSize The number of blocks in the blockchain  
 \* @param difficulty The difficulty of the most recent block  
 \* @param totalDifficulty The total difficulty across all blocks  
 \* @param hashesPerSecond The blockchain's approximate hash rate  
 \* @param totalExpectedHashes The expected total number of hashes for the entire blockchain  
 \* @param nonce The nonce of the last block  
 \* @param startTime The start time of the operation  
 \* @param endTime The end time of the operation  
 \*/* public ResponseMessage(String status, String message, String chainHash, int chainSize, int difficulty, int totalDifficulty, int hashesPerSecond, double totalExpectedHashes, BigInteger nonce, long startTime, long endTime) {  
 this.status = status;  
 this.message = message;  
 this.chainHash = chainHash;  
 this.chainSize = chainSize;  
 this.difficulty = difficulty;  
 this.totalDifficulty = totalDifficulty;  
 this.hashesPerSecond = hashesPerSecond;  
 this.totalExpectedHashes = totalExpectedHashes;  
 this.nonce = nonce;  
 this.startTime = startTime;  
 this.endTime = endTime;  
 }  
  
 */\*\*  
 \* Gets the response status.  
 \*  
 \* @return The status of the response  
 \*/* public String getStatus() {  
 return status;  
 }  
  
 */\*\*  
 \* Gets the message associated with the response.  
 \*  
 \* @return The message detailing the response  
 \*/* public String getMessage() {  
 return message;  
 }  
  
 */\*\*  
 \* Gets the hash of the blockchain.  
 \*  
 \* @return The current chain hash  
 \*/* public String getChainHash() {  
 return chainHash;  
 }  
  
 */\*\*  
 \* Gets the number of blocks in the blockchain.  
 \*  
 \* @return The total number of blocks  
 \*/* public int getChainSize() {  
 return chainSize;  
 }  
  
 */\*\*  
 \* Gets the difficulty level of the last block in the blockchain.  
 \*  
 \* @return The difficulty of the last block  
 \*/* public int getDifficulty() {  
 return difficulty;  
 }  
  
 */\*\*  
 \* Gets the total difficulty of the blockchain.  
 \*  
 \* @return The aggregated difficulty of all blocks  
 \*/* public int getTotalDifficulty() {  
 return totalDifficulty;  
 }  
  
 */\*\*  
 \* Gets the approximate hash rate of the system.  
 \*  
 \* @return The hash rate in hashes per second  
 \*/* public int getHashesPerSecond() {  
 return hashesPerSecond;  
 }  
  
 */\*\*  
 \* Gets the total expected number of hashes for the blockchain.  
 \*  
 \* @return The total expected hashes based on difficulty  
 \*/* public double getTotalExpectedHashes() {  
 return totalExpectedHashes;  
 }  
  
 */\*\*  
 \* Gets the nonce of the last block.  
 \*  
 \* @return The nonce of the most recent block  
 \*/* public BigInteger getNonce() {  
 return nonce;  
 }  
  
 */\*\*  
 \* Gets the start time of the operation.  
 \*  
 \* @return The operation start timestamp  
 \*/* public long getStartTime() {  
 return startTime;  
 }  
  
 */\*\*  
 \* Gets the end time of the operation.  
 \*  
 \* @return The operation end timestamp  
 \*/* public long getEndTime() {  
 return endTime;  
 }  
  
 */\*\*  
 \* Provides a JSON-like string representation of the response message.  
 \*  
 \* @return A string representation of the response message  
 \*/* @Override  
 public String toString() {  
 return "{\"status\":\"" + status + "\", \"message\":\"" + message + "\", \"chainHash\":\"" + chainHash + "\", \"chainSize\":" + chainSize +  
 ", \"difficulty\":" + difficulty + ", \"totalDifficulty\":" + totalDifficulty + ", \"hashesPerSecond\":" + hashesPerSecond +  
 ", \"totalExpectedHashes\":" + totalExpectedHashes + ", \"nonce\":" + nonce + "}";  
 }  
}

**Separation of concerns:**

public void addBlock(String transactionData, int difficultyLevel) {  
 long startTime = System.*currentTimeMillis*();  
 Block newBlock = new Block(blockchain.size(), new Timestamp(System.*currentTimeMillis*()), transactionData, difficultyLevel);  
 newBlock.setPreviousHash(chainHash);  
 newBlock.mineBlock();  
 chainHash = newBlock.calculateHash();  
 blockchain.add(newBlock);  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.printf("Total execution time to add this block was %d milliseconds%n", (endTime - startTime));  
}

The above method keeps the process of adding blocks separate from how individual blocks are structured or validated.

public String isChainValid() {  
 long startTime = System.*currentTimeMillis*();  
 String previousHash = "0";  
  
 for (int i = 0; i < blockchain.size(); i++) {  
 Block currentBlock = blockchain.get(i);  
 String calculatedHash = currentBlock.calculateHash();  
  
 // Check if the previous hash matches the stored previous hash  
 if (!currentBlock.getPreviousHash().equals(previousHash)) {  
 System.*out*.println("Chain verification: FALSE");  
 System.*out*.printf("Improper hash on node %d. Does not match previous hash.%n", i);  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.printf("Total execution time required to verify the chain was %d milliseconds%n", (endTime - startTime));  
 return "FALSE"; // Return as a string indicating the chain is invalid  
 }  
 previousHash = calculatedHash; // Update previous hash to current for the next iteration  
 }  
  
 // If no issues found, return "TRUE"  
 return "TRUE";  
}

This method focuses solely on validation logic, not on how blocks are created or stored.

**Task 1 ClientSideExecution:**/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=59804:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task1/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.ClientTCP

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Experimented with 2,000,000 hashes.

Approximate hashes per second on this machine: 666666666

Expected total hashes required for the whole chain: 256.0

Nonce for most recent block: 53

Chain hash: 00ef9364cafd7d928021c1317ae5d320699de01b25693a3e66402c7abc5c5f47

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Alice pays Bob 100 DSCoin

Total execution time to add this block was 278 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Bob pays Carol 20 DSCoin

Total execution time to add this block was 107 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Carol pays Donna 10 DSCoin

Total execution time to add this block was 39 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 08:43:45.04","Tx ": "Genesis","PrevHash" : "0","nonce" : 53,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 08:44:05.633","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00ef9364cafd7d928021c1317ae5d320699de01b25693a3e66402c7abc5c5f47","nonce" : 127589,"difficulty": 4},{"index" : 2,"time stamp " : "2024-10-31 08:44:19.071","Tx ": "Bob pays Carol 20 DSCoin","PrevHash" : "00007bb305a006abad0a19737a27d1033765059ef53386788a35ef040479bbff","nonce" : 47402,"difficulty": 4},{"index" : 3,"time stamp " : "2024-10-31 08:44:28.446","Tx ": "Carol pays Donna 10 DSCoin","PrevHash" : "00002fa8f79efa10d3125a40ef4e798f2bd635a6f1b55f5cb0370975a1373e41","nonce" : 12612,"difficulty": 4}], "chainHash":"0000ee3c222c9c69469451414d733f8b54fe9cc666e6af154f5209ff3c18f4cf"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: TRUE

Total execution time required to verify the chain was 1 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 4

Corrupt the Blockchain

Enter block ID of block to corrupt: 2

Enter new data for block 2: Bob pays Tony 30 DSCoin

Block 2 now holds Bob pays Tony 30 DSCoin

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: FALSE

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 5

Repairing the entire chain

Total execution time required to repair the chain was 652 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Verifying entire chain

Chain verification: TRUE

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 08:43:45.04","Tx ": "Genesis","PrevHash" : "0","nonce" : 53,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 08:44:05.633","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00ef9364cafd7d928021c1317ae5d320699de01b25693a3e66402c7abc5c5f47","nonce" : 127589,"difficulty": 4},{"index" : 2,"time stamp " : "2024-10-31 08:44:19.071","Tx ": "Bob pays Tony 30 DSCoin","PrevHash" : "00007bb305a006abad0a19737a27d1033765059ef53386788a35ef040479bbff","nonce" : 91926,"difficulty": 4},{"index" : 3,"time stamp " : "2024-10-31 08:44:28.446","Tx ": "Carol pays Donna 10 DSCoin","PrevHash" : "0000f3f933ba2a30f9c0ff89fb38128e082ec7bbf9ea21a03807a786b596284f","nonce" : 443180,"difficulty": 4}], "chainHash":"0000f9ab713f23e07342ae9bad0ede9f67a355c982f046c7df7ba61901b56176"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 6

Exiting...

Process finished with exit code 0

**Task 1 Server Side Execution:**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=59801:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task1/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.ServerTCP

Blockchain Server Running

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"getStatus","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing status","chainHash":"00ef9364cafd7d928021c1317ae5d320699de01b25693a3e66402c7abc5c5f47","chainSize":1,"difficulty":2,"totalDifficulty":2,"hashesPerSecond":666666666,"totalExpectedHashes":256.0,"nonce":53,"startTime":1730378631381,"endTime":1730378631381}

Number of Blocks on Chain: 1

Total execution time to add this block was 276 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"addBlock","transactionData":"Alice pays Bob 100 DSCoin","difficultyOrIndex":4}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"00007bb305a006abad0a19737a27d1033765059ef53386788a35ef040479bbff","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":666666666,"totalExpectedHashes":65792.0,"nonce":127589,"startTime":1730378645632,"endTime":1730378645910}

Number of Blocks on Chain: 2

Total execution time to add this block was 107 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"addBlock","transactionData":"Bob pays Carol 20 DSCoin","difficultyOrIndex":4}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"00002fa8f79efa10d3125a40ef4e798f2bd635a6f1b55f5cb0370975a1373e41","chainSize":3,"difficulty":4,"totalDifficulty":10,"hashesPerSecond":666666666,"totalExpectedHashes":131328.0,"nonce":47402,"startTime":1730378659071,"endTime":1730378659178}

Number of Blocks on Chain: 3

Total execution time to add this block was 38 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"addBlock","transactionData":"Carol pays Donna 10 DSCoin","difficultyOrIndex":4}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"0000ee3c222c9c69469451414d733f8b54fe9cc666e6af154f5209ff3c18f4cf","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":12612,"startTime":1730378668446,"endTime":1730378668485}

Number of Blocks on Chain: 4

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"viewChain","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing blockchain","chainHash":"{\"ds\_chain\" : [{\"index\" : 0,\"time stamp \" : \"2024-10-31 08:43:45.04\",\"Tx \": \"Genesis\",\"PrevHash\" : \"0\",\"nonce\" : 53,\"difficulty\": 2},{\"index\" : 1,\"time stamp \" : \"2024-10-31 08:44:05.633\",\"Tx \": \"Alice pays Bob 100 DSCoin\",\"PrevHash\" : \"00ef9364cafd7d928021c1317ae5d320699de01b25693a3e66402c7abc5c5f47\",\"nonce\" : 127589,\"difficulty\": 4},{\"index\" : 2,\"time stamp \" : \"2024-10-31 08:44:19.071\",\"Tx \": \"Bob pays Carol 20 DSCoin\",\"PrevHash\" : \"00007bb305a006abad0a19737a27d1033765059ef53386788a35ef040479bbff\",\"nonce\" : 47402,\"difficulty\": 4},{\"index\" : 3,\"time stamp \" : \"2024-10-31 08:44:28.446\",\"Tx \": \"Carol pays Donna 10 DSCoin\",\"PrevHash\" : \"00002fa8f79efa10d3125a40ef4e798f2bd635a6f1b55f5cb0370975a1373e41\",\"nonce\" : 12612,\"difficulty\": 4}], \"chainHash\":\"0000ee3c222c9c69469451414d733f8b54fe9cc666e6af154f5209ff3c18f4cf\"}","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":12612,"startTime":1730378673378,"endTime":1730378673378}

Number of Blocks on Chain: 4

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"verifyChain","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"TRUE","chainHash":"0000ee3c222c9c69469451414d733f8b54fe9cc666e6af154f5209ff3c18f4cf","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":12612,"startTime":1730378678466,"endTime":1730378678467}

Number of Blocks on Chain: 4

Block 2 now holds Bob pays Tony 30 DSCoin

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"corruptBlock","transactionData":"Bob pays Tony 30 DSCoin","difficultyOrIndex":2}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block 2 corrupted with new data","chainHash":"0000ee3c222c9c69469451414d733f8b54fe9cc666e6af154f5209ff3c18f4cf","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":12612,"startTime":1730378699368,"endTime":1730378699369}

Number of Blocks on Chain: 4

Chain verification: FALSE

Improper hash on node 3. Does not match previous hash.

Total execution time required to verify the chain was 0 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"verifyChain","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"FALSE","chainHash":"0000ee3c222c9c69469451414d733f8b54fe9cc666e6af154f5209ff3c18f4cf","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":12612,"startTime":1730378704682,"endTime":1730378704682}

Number of Blocks on Chain: 4

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"repairChain","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Blockchain repaired","chainHash":"0000f9ab713f23e07342ae9bad0ede9f67a355c982f046c7df7ba61901b56176","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":443180,"startTime":1730378717184,"endTime":1730378717836}

Number of Blocks on Chain: 4

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"verifyChain","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"TRUE","chainHash":"0000f9ab713f23e07342ae9bad0ede9f67a355c982f046c7df7ba61901b56176","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":443180,"startTime":1730378726417,"endTime":1730378726417}

Number of Blocks on Chain: 4

We have a visitor

THE JSON REQUEST MESSAGE IS: {"action":"viewChain","difficultyOrIndex":0}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing blockchain","chainHash":"{\"ds\_chain\" : [{\"index\" : 0,\"time stamp \" : \"2024-10-31 08:43:45.04\",\"Tx \": \"Genesis\",\"PrevHash\" : \"0\",\"nonce\" : 53,\"difficulty\": 2},{\"index\" : 1,\"time stamp \" : \"2024-10-31 08:44:05.633\",\"Tx \": \"Alice pays Bob 100 DSCoin\",\"PrevHash\" : \"00ef9364cafd7d928021c1317ae5d320699de01b25693a3e66402c7abc5c5f47\",\"nonce\" : 127589,\"difficulty\": 4},{\"index\" : 2,\"time stamp \" : \"2024-10-31 08:44:19.071\",\"Tx \": \"Bob pays Tony 30 DSCoin\",\"PrevHash\" : \"00007bb305a006abad0a19737a27d1033765059ef53386788a35ef040479bbff\",\"nonce\" : 91926,\"difficulty\": 4},{\"index\" : 3,\"time stamp \" : \"2024-10-31 08:44:28.446\",\"Tx \": \"Carol pays Donna 10 DSCoin\",\"PrevHash\" : \"0000f3f933ba2a30f9c0ff89fb38128e082ec7bbf9ea21a03807a786b596284f\",\"nonce\" : 443180,\"difficulty\": 4}], \"chainHash\":\"0000f9ab713f23e07342ae9bad0ede9f67a355c982f046c7df7ba61901b56176\"}","chainSize":4,"difficulty":4,"totalDifficulty":14,"hashesPerSecond":666666666,"totalExpectedHashes":196864.0,"nonce":443180,"startTime":1730378729661,"endTime":1730378729661}

Number of Blocks on Chain: 4

Client disconnected

Process finished with exit code 130 (interrupted by signal 2: SIGINT)

**Project3Task2SigningClient**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
package org.example;  
  
import com.google.gson.Gson;  
import java.security.MessageDigest;  
import java.io.\*;  
import java.math.BigInteger;  
import java.net.Socket;  
import java.security.NoSuchAlgorithmException;  
import java.util.Random;  
  
*/\*\* \* SigningClientTCP serves as a client to interact with the blockchain server over TCP sockets. \* It generates RSA keys, sends signed JSON-formatted requests for blockchain operations, \* such as adding a block, verifying the chain, viewing the blockchain, and repairing the chain. \*/*

public class SigningClientTCP {  
 public static void main(String[] args) {  
 Socket clientSocket = null;  
 try {  
 // Connect to the server socket running on localhost:7777  
 int serverPort = 7777;  
 clientSocket = new Socket("localhost", serverPort);  
  
 // Set up input and output streams for communication with the server  
 BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
 PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));  
  
 // Set up Gson for JSON serialization/deserialization  
 Gson gson = new Gson();  
 BufferedReader typed = new BufferedReader(new InputStreamReader(System.*in*));  
  
 // RSA key generation  
 Random rnd = new Random();  
 BigInteger p = BigInteger.*probablePrime*(400, rnd); // Generate a 400-bit prime number  
 BigInteger q = BigInteger.*probablePrime*(400, rnd); // Generate another 400-bit prime number  
 BigInteger n = p.multiply(q); // Calculate n = p \* q  
 BigInteger phi = p.subtract(BigInteger.*ONE*).multiply(q.subtract(BigInteger.*ONE*)); // Calculate phi  
 BigInteger e = BigInteger.*valueOf*(65537); // Public exponent  
 BigInteger d = e.modInverse(phi); // Private exponent  
  
 // Display public and private keys  
 System.*out*.println("Public key of client (e, n): (" + e + ", " + n + ")");  
 System.*out*.println("Private key of client (d, n): (" + d + ", " + n + ")");  
  
 // Compute client ID from public key  
 String publicKeyConcat = e.toString() + n.toString(); // Concatenate public key components  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256"); // SHA-256 hash algorithm  
 byte[] publicKeyHash = md.digest(publicKeyConcat.getBytes()); // Hash the public key  
 String clientId = new BigInteger(1, publicKeyHash).toString(16).substring(publicKeyHash.length - 20); // Generate client ID  
 System.*out*.println("Client id: " + clientId); // Display client ID  
  
 int choice;  
 int blockId = 0; // Declare blockId for corrupting specific blocks  
  
 do {  
 // Display menu options  
 System.*out*.println("\n0. View basic blockchain status.");  
 System.*out*.println("1. Add a transaction to the blockchain.");  
 System.*out*.println("2. Verify the blockchain.");  
 System.*out*.println("3. View the blockchain.");  
 System.*out*.println("4. Corrupt the chain.");  
 System.*out*.println("5. Hide the corruption by repairing the chain.");  
 System.*out*.println("6. Exit");  
 System.*out*.print("Enter your choice: ");  
 String choiceInput = typed.readLine().trim();  
 choice = Integer.*parseInt*(choiceInput);  
  
 RequestMessage requestMessage = null;  
 switch (choice) {  
 case 0: // Retrieve blockchain status  
 requestMessage = new RequestMessage("getStatus", null, 0);  
 break;  
 case 1: // Add a new block to the blockchain  
 System.*out*.print("Enter difficulty > 1: ");  
 int difficulty = Integer.*parseInt*(typed.readLine());  
 System.*out*.print("Enter transaction: ");  
 String transaction = typed.readLine();  
 requestMessage = new RequestMessage("addBlock", transaction, difficulty);  
 break;  
 case 2: // Verify the integrity of the blockchain  
 requestMessage = new RequestMessage("verifyChain", null, 0);  
 break;  
 case 3: // View the entire blockchain  
 requestMessage = new RequestMessage("viewChain", null, 0);  
 break;  
 case 4: // Corrupt a block in the blockchain  
 System.*out*.println("Corrupt the Blockchain");  
 System.*out*.print("Enter block ID of block to corrupt: ");  
 blockId = Integer.*parseInt*(typed.readLine()); // Initialize blockId here  
 System.*out*.print("Enter new data for block " + blockId + ": ");  
 String newData = typed.readLine();  
 requestMessage = new RequestMessage("corruptBlock", newData, blockId);  
 break;  
 case 5: // Repair the blockchain  
 requestMessage = new RequestMessage("repairChain", null, 0);  
 break;  
 case 6: // Exit  
 System.*out*.println("Exiting...");  
 break;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 break;  
 }  
  
 // Send the request to the server and process the response  
 if (requestMessage != null) {  
 // Prepare the data to sign  
 String requestString = gson.toJson(requestMessage); // Convert request to JSON  
 byte[] requestHash = md.digest(requestString.getBytes()); // Hash the request  
 BigInteger requestHashBigInt = new BigInteger(1, requestHash); // Ensure the hash is positive  
  
 // Sign the request  
 BigInteger signature = requestHashBigInt.modPow(d, n); // Sign the request  
  
 // Create the signed request  
 SignedRequest authenticatedRequest = new SignedRequest(requestMessage, signature.toString(), e.toString(), n.toString(), clientId); // Create signed request  
 String jsonRequest = gson.toJson(authenticatedRequest); // Convert signed request to JSON  
  
 // Send request to the server  
 out.println(jsonRequest);  
 out.flush(); // Flush output stream  
 String jsonResponse = in.readLine(); // Read response from server  
 ResponseMessage responseMessage = gson.fromJson(jsonResponse, ResponseMessage.class); // Parse JSON response  
  
 // Process the server response  
 switch (choice) {  
 case 0: // Display blockchain status  
 System.*out*.println("Current size of chain: " + responseMessage.getChainSize());  
 System.*out*.println("Difficulty of most recent block: " + responseMessage.getDifficulty());  
 System.*out*.println("Total difficulty for all blocks: " + responseMessage.getTotalDifficulty());  
 System.*out*.println("Approximate hashes per second on this machine: " + responseMessage.getHashesPerSecond());  
 System.*out*.println("Expected total hashes required for the whole chain: " + responseMessage.getTotalExpectedHashes());  
 System.*out*.println("Nonce for most recent block: " + responseMessage.getNonce());  
 System.*out*.println("Chain hash: " + responseMessage.getChainHash());  
 break;  
 case 1: // Display block creation time  
 long blockCreationTime = responseMessage.getEndTime() - responseMessage.getStartTime();  
 System.*out*.println("Total execution time to add this block was " + blockCreationTime + " milliseconds");  
 break;  
 case 2: // Display chain verification result  
 long verificationTime = responseMessage.getEndTime() - responseMessage.getStartTime();  
 System.*out*.println("Chain verification: " + responseMessage.getMessage());  
 System.*out*.println("Total execution time required to verify the chain was " + verificationTime + " milliseconds");  
 break;  
 case 3: // Display the blockchain  
 System.*out*.println("View the Blockchain");  
 System.*out*.println(responseMessage.getChainHash());  
 break;  
 case 4: // Display confirmation of block corruption  
 System.*out*.println("Block " + blockId + " now holds " + requestMessage.getTransactionData());  
 break;  
 case 5: // Display chain repair time  
 long repairTime = responseMessage.getEndTime() - responseMessage.getStartTime();  
 System.*out*.println("Total execution time required to repair the chain was " + repairTime + " milliseconds");  
 break;  
 case 6: // Exit  
 break;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 break;  
 }  
 }  
 } while (choice != 6);  
 } catch (IOException | NoSuchAlgorithmException e) {  
 System.*out*.println("Error: " + e.getMessage());  
 } finally {  
 try {  
 if (clientSocket != null) {  
 clientSocket.close();  
 }  
 } catch (IOException e) {  
 // Ignore exception on close  
 }  
 }  
 }  
}

**Project3Task2VerifyingServerTCP**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \*/  
package org.example;  
  
import com.google.gson.Gson;  
  
import java.io.BufferedWriter;  
import java.io.IOException;  
import java.io.OutputStreamWriter;  
import java.io.PrintWriter;  
import java.math.BigInteger;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.security.MessageDigest;  
import java.sql.Timestamp;  
import java.util.Scanner;  
  
*/\*\**

*\* VerifyingServerTCP handles client requests over TCP sockets, verifies signed requests,*

*\* manages the blockchain on the server, and processes various blockchain operations.*

*\*/*

public class VerifyingServerTCP {  
 public static void main(String[] args) {  
 try {  
 // Set up server socket to listen on port 7777  
 int port = 7777;  
 ServerSocket serverSocket = new ServerSocket(port);  
 System.*out*.println("Blockchain Server Running");  
  
 // Initialize blockchain with a genesis block  
 Blockchain blockchain = new Blockchain();  
  
 // Continuously listen for incoming client connections  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 try (Scanner inputScanner = new Scanner(clientSocket.getInputStream());  
 PrintWriter outputWriter = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())))) {  
  
 Gson gson = new Gson();  
  
 while (inputScanner.hasNextLine()) {  
 // Process each incoming client request  
 String requestJson = inputScanner.nextLine();  
 SignedRequest signedRequest = gson.fromJson(requestJson, SignedRequest.class);  
 RequestMessage request = signedRequest.getTransactionRequest();  
  
 // Verify client ID and signature  
 if (!*isValidRequest*(signedRequest)) {  
 outputWriter.println(gson.toJson(new ResponseMessage("error", "Invalid request", null, -1, 0, 0, 0, 0, null, 0L, 0L)));  
 outputWriter.flush();  
 continue;  
 }  
  
 String responseJson = *processRequest*(request, blockchain, gson);  
 outputWriter.println(responseJson);  
 outputWriter.flush();  
  
 // Log request and response details for debugging  
 System.*out*.println("\nWe have a visitor");  
 System.*out*.println("THE JSON REQUEST MESSAGE IS: " + requestJson);  
 System.*out*.println("THE JSON RESPONSE MESSAGE IS: " + responseJson);  
 System.*out*.println("Number of Blocks on Chain: " + blockchain.getSize());  
 }  
 } catch (IOException e) {  
 System.*out*.println("Error handling client: " + e.getMessage());  
 } finally {  
 System.*out*.println("Client disconnected");  
 clientSocket.close();  
 }  
 }  
 } catch (IOException e) {  
 System.*out*.println("Server error: " + e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* Validates the client's request by checking the public key and verifying the signature.  
 \*  
 \* @param signedRequest The signed request object from the client.  
 \* @return true if the request is valid, false otherwise.  
 \*/* private static boolean isValidRequest(SignedRequest signedRequest) {  
 try {  
 // Recreate the hash of the request for signature verification  
 Gson gson = new Gson();  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 String requestString = gson.toJson(signedRequest.getTransactionRequest()); // Convert request to JSON  
 byte[] requestHash = md.digest(requestString.getBytes()); // Hash the request  
 BigInteger requestHashBigInt = new BigInteger(1, requestHash); // Ensure the hash is positive  
  
 // Get the public key components from the signed request  
 BigInteger e = new BigInteger(signedRequest.getPublicKeyExponent());  
 BigInteger n = new BigInteger(signedRequest.getPublicKeyModulus());  
  
 // Verify the client's ID  
 String publicKeyConcat = signedRequest.getPublicKeyExponent() + signedRequest.getPublicKeyModulus();  
 MessageDigest idDigest = MessageDigest.*getInstance*("SHA-256");  
 byte[] idHash = idDigest.digest(publicKeyConcat.getBytes());  
 String clientId = new BigInteger(1, idHash).toString(16).substring(idHash.length - 20);  
  
 if (!clientId.equals(signedRequest.getClientIdentifier())) {  
 return false; // Invalid public key ID  
 }  
  
 // Decrypt the signature using the public key  
 BigInteger decryptedSignature = new BigInteger(signedRequest.getDigitalSignature()).modPow(e, n);  
  
 // Check if the recreated hash matches the decrypted signature  
 boolean isValid = requestHashBigInt.equals(decryptedSignature);  
 // Print verification status  
 if (isValid) {  
 System.*out*.println("Signature verified successfully.");  
 } else {  
 System.*out*.println("Signature verification failed.");  
 }  
 return isValid;  
 } catch (Exception e) {  
 System.*err*.println("Error in request validation: " + e.getMessage());  
 return false; // Return false in case of any error  
 }  
 }  
  
 */\*\*  
 \* Processes incoming requests and generates appropriate responses.  
 \*  
 \* @param request The request message received from the client.  
 \* @param blockchain The blockchain instance to perform operations on.  
 \* @param gson Gson instance for JSON processing.  
 \* @return JSON-formatted response message.  
 \*/* private static String processRequest(RequestMessage request, Blockchain blockchain, Gson gson) {  
 long startTime, endTime;  
 String responseJson;  
  
 switch (request.getAction()) {  
 case "getStatus": // Retrieve blockchain status  
 startTime = System.*currentTimeMillis*();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Viewing status", blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "addBlock": // Add a new block to the blockchain  
 startTime = System.*currentTimeMillis*();  
 Block newBlock = new Block(blockchain.getSize(), new Timestamp(System.*currentTimeMillis*()), request.getTransactionData(), request.getDifficultyOrIndex());  
 blockchain.addBlock(request.getTransactionData(), request.getDifficultyOrIndex());  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Block added", blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "verifyChain": // Verify the blockchain's integrity  
 startTime = System.*currentTimeMillis*();  
 String validationResult = blockchain.isChainValid();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", validationResult, blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "viewChain": // View the entire blockchain  
 startTime = System.*currentTimeMillis*();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Viewing blockchain", blockchain.toString(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 case "corruptBlock": // Corrupt a specific block in the blockchain  
 startTime = System.*currentTimeMillis*();  
 int blockIndex = request.getDifficultyOrIndex();  
 if (blockIndex >= 0 && blockIndex < blockchain.getSize()) {  
 blockchain.corruptBlock(blockIndex, request.getTransactionData());  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Block " + blockIndex + " corrupted with new data",  
 blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 } else {  
 responseJson = gson.toJson(new ResponseMessage(  
 "error", "Invalid block index", null, blockchain.getSize(),  
 0, blockchain.getTotalDifficulty(), 0, 0, null, 0L, 0L));  
 }  
 break;  
  
 case "repairChain": // Repair the blockchain by recalculating hashes  
 startTime = System.*currentTimeMillis*();  
 blockchain.repairChain();  
 endTime = System.*currentTimeMillis*();  
 responseJson = gson.toJson(new ResponseMessage(  
 "success", "Blockchain repaired", blockchain.getBlockchainHash(), blockchain.getSize(),  
 blockchain.getLastBlock().getDifficulty(), blockchain.getTotalDifficulty(),  
 blockchain.getHashRate(), blockchain.getExpectedTotalHashes(),  
 blockchain.getLastBlock().getNonce(), startTime, endTime));  
 break;  
  
 default: // Invalid action  
 responseJson = gson.toJson(new ResponseMessage(  
 "error", "Invalid action", null, -1, 0, 0, 0, 0, null, 0L, 0L));  
 break;  
 }  
 return responseJson;  
 }  
}

**SignedRequest.java**

/\*  
 \* Khushi Bhuwania  
 \* kbhuwani@andrew.cmu.edu  
 \* \*/  
  
package org.example;  
  
*/\*\*  
 \* The SignedRequest class represents a signed request from a client to the server.  
 \* It includes the transaction request, digital signature, public key components,  
 \* and a unique client identifier for verification purposes.  
 \*/*public class SignedRequest {  
 /\* The request message containing the transaction details \*/  
 private RequestMessage transactionRequest;  
  
 /\* The digital signature of the request, used to verify authenticity \*/  
 private String digitalSignature;  
  
 /\* The public key exponent used for verifying the signature \*/  
 private String publicKeyExponent;  
  
 /\* The public key modulus used for verifying the signature \*/  
 private String publicKeyModulus;  
  
 /\* The identifier of the client, derived from the client's public key \*/  
 private String clientIdentifier;  
  
 */\*\*  
 \* Constructor for SignedRequest.  
 \*  
 \* @param transactionRequest The request message to be signed.  
 \* @param digitalSignature The digital signature of the request.  
 \* @param publicKeyExponent The public key exponent used for signature verification.  
 \* @param publicKeyModulus The public key modulus used for signature verification.  
 \* @param clientIdentifier The unique identifier of the client.  
 \*/* public SignedRequest(RequestMessage transactionRequest, String digitalSignature,  
 String publicKeyExponent, String publicKeyModulus,  
 String clientIdentifier) {  
 this.transactionRequest = transactionRequest; // Assign the transaction request  
 this.digitalSignature = digitalSignature; // Assign the digital signature  
 this.publicKeyExponent = publicKeyExponent; // Assign the public key exponent  
 this.publicKeyModulus = publicKeyModulus; // Assign the public key modulus  
 this.clientIdentifier = clientIdentifier; // Assign the client identifier  
 }  
  
 // Getter for the transaction request  
 public RequestMessage getTransactionRequest() {  
 return transactionRequest; // Return the transaction request  
 }  
  
 // Getter for the digital signature  
 public String getDigitalSignature() {  
 return digitalSignature; // Return the digital signature  
 }  
  
 // Getter for the public key exponent  
 public String getPublicKeyExponent() {  
 return publicKeyExponent; // Return the public key exponent  
 }  
  
 // Getter for the public key modulus  
 public String getPublicKeyModulus() {  
 return publicKeyModulus; // Return the public key modulus  
 }  
  
 // Getter for the client identifier  
 public String getClientIdentifier() {  
 return clientIdentifier; // Return the client identifier  
 }  
  
 */\*\*  
 \* Override the toString method for printing the SignedRequest.  
 \*  
 \* @return A string representation of the SignedRequest object.  
 \*/* @Override  
 public String toString() {  
 return "SignedRequest{" +  
 "transactionRequest=" + transactionRequest + // Print the transaction request  
 ", digitalSignature='" + digitalSignature + '\'' + // Print the digital signature  
 ", publicKeyExponent='" + publicKeyExponent + '\'' + // Print the public key exponent  
 ", publicKeyModulus='" + publicKeyModulus + '\'' + // Print the public key modulus  
 ", clientIdentifier='" + clientIdentifier + '\'' + // Print the client identifier  
 '}';  
 }  
}

**Project3 Task2 Server and Client Console Outputs**

**Output: Client 1**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=62390:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task2/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.SigningClientTCP

Public key of client (e, n): (65537, 4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611)

Private key of client (d, n): (4126224064407766765692100342018572762064013694357464783106050372555589511074145971924144186730163068998225437859187410755378271107378091356005828565277251137262139454965895564354489496574376036329335629933176391525372264437955880553992181249, 4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611)

Client id: 96fe3031094a680489bd6223b73def9bff86a494a77b3046a996

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Approximate hashes per second on this machine: 666666666

Expected total hashes required for the whole chain: 256.0

Nonce for most recent block: 17

Chain hash: 00e43cb9332d609d9162179401d819121ea86dcad1e11cfd6a1db05a012dfeaf

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Alice pays Bob 100 DSCoin

Total execution time to add this block was 341 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Bob pays Carol 20 DSCoin

Total execution time to add this block was 49 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Chain verification: TRUE

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 11:57:12.171","Tx ": "Genesis","PrevHash" : "0","nonce" : 17,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 12:40:14.954","Tx ": "Alice pays Bob 100 DSCoin","PrevHash" : "00e43cb9332d609d9162179401d819121ea86dcad1e11cfd6a1db05a012dfeaf","nonce" : 156453,"difficulty": 4},{"index" : 2,"time stamp " : "2024-10-31 12:41:11.538","Tx ": "Bob pays Carol 20 DSCoin","PrevHash" : "0000904079465ea84586da7f8c11f4d1bba7a3fea403ee46e896325898f04e41","nonce" : 16100,"difficulty": 4}], "chainHash":"000074bbfb84e7e366ea8f127df3650701f985a15a22519e31fba304d1c7c87a"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 6

Exiting...

Process finished with exit code 0

**Output: Server 1**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=62387:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task2/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.VerifyingServerTCP

Blockchain Server Running

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"getStatus","difficultyOrIndex":0},"digitalSignature":"4050147053074492385513427786830066722771658675202214638868133989065466387946078289122871070003699030505674608915250297202709566065838253151817393351902333843143202084787173675087489413550267960025680757391683805149123393533130608294832682333","publicKeyExponent":"65537","publicKeyModulus":"4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611","clientIdentifier":"96fe3031094a680489bd6223b73def9bff86a494a77b3046a996"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing status","chainHash":"00e43cb9332d609d9162179401d819121ea86dcad1e11cfd6a1db05a012dfeaf","chainSize":1,"difficulty":2,"totalDifficulty":2,"hashesPerSecond":666666666,"totalExpectedHashes":256.0,"nonce":17,"startTime":1730390239322,"endTime":1730390239322}

Number of Blocks on Chain: 1

Signature verified successfully.

Total execution time to add this block was 341 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"addBlock","transactionData":"Alice pays Bob 100 DSCoin","difficultyOrIndex":4},"digitalSignature":"4109879476972263059407014642876294154453084867852390613103977241481506101162345140376804365485870763624203917120967972959324506541801669770480974881018763529193574216152560263660218754546767339580303936417489663612504283267532781388048137281","publicKeyExponent":"65537","publicKeyModulus":"4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611","clientIdentifier":"96fe3031094a680489bd6223b73def9bff86a494a77b3046a996"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"0000904079465ea84586da7f8c11f4d1bba7a3fea403ee46e896325898f04e41","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":666666666,"totalExpectedHashes":65792.0,"nonce":156453,"startTime":1730392814954,"endTime":1730392815295}

Number of Blocks on Chain: 2

Signature verified successfully.

Total execution time to add this block was 49 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"addBlock","transactionData":"Bob pays Carol 20 DSCoin","difficultyOrIndex":4},"digitalSignature":"542293585891171833314360221828417881820905571169312908193917937513514419213498214676623329778249784193739105094751210175476045166840361530962140114682998766017597849996513055680039175984715996284083931725970272696618943867213190946260296586","publicKeyExponent":"65537","publicKeyModulus":"4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611","clientIdentifier":"96fe3031094a680489bd6223b73def9bff86a494a77b3046a996"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"000074bbfb84e7e366ea8f127df3650701f985a15a22519e31fba304d1c7c87a","chainSize":3,"difficulty":4,"totalDifficulty":10,"hashesPerSecond":666666666,"totalExpectedHashes":131328.0,"nonce":16100,"startTime":1730392871538,"endTime":1730392871587}

Number of Blocks on Chain: 3

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"verifyChain","difficultyOrIndex":0},"digitalSignature":"2534655987933129007895363585447553195255343488396800125442063952778232959520730178968821678579262017536543213570670912859893375395449765473605943883351421437015258770842794007794277601281506260727288436858864618158356955552883833826909448828","publicKeyExponent":"65537","publicKeyModulus":"4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611","clientIdentifier":"96fe3031094a680489bd6223b73def9bff86a494a77b3046a996"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"TRUE","chainHash":"000074bbfb84e7e366ea8f127df3650701f985a15a22519e31fba304d1c7c87a","chainSize":3,"difficulty":4,"totalDifficulty":10,"hashesPerSecond":666666666,"totalExpectedHashes":131328.0,"nonce":16100,"startTime":1730392873629,"endTime":1730392873629}

Number of Blocks on Chain: 3

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"viewChain","difficultyOrIndex":0},"digitalSignature":"3625436077205474356678419692456970697950505190042349746741107339272887332113051696104703055735921634436491939369112521833063162562310561018242934027813789925395065046533466931829972073209584332718461226872758641488258456015664159956346212328","publicKeyExponent":"65537","publicKeyModulus":"4421089273601213264283477423975267356168284103703122151038505432203767939496882329431263080236319148758079660611737980886994595378781043628144775294931979313607732291543215799126338026108201624180688331330627139564800347865599894217515704611","clientIdentifier":"96fe3031094a680489bd6223b73def9bff86a494a77b3046a996"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing blockchain","chainHash":"{\"ds\_chain\" : [{\"index\" : 0,\"time stamp \" : \"2024-10-31 11:57:12.171\",\"Tx \": \"Genesis\",\"PrevHash\" : \"0\",\"nonce\" : 17,\"difficulty\": 2},{\"index\" : 1,\"time stamp \" : \"2024-10-31 12:40:14.954\",\"Tx \": \"Alice pays Bob 100 DSCoin\",\"PrevHash\" : \"00e43cb9332d609d9162179401d819121ea86dcad1e11cfd6a1db05a012dfeaf\",\"nonce\" : 156453,\"difficulty\": 4},{\"index\" : 2,\"time stamp \" : \"2024-10-31 12:41:11.538\",\"Tx \": \"Bob pays Carol 20 DSCoin\",\"PrevHash\" : \"0000904079465ea84586da7f8c11f4d1bba7a3fea403ee46e896325898f04e41\",\"nonce\" : 16100,\"difficulty\": 4}], \"chainHash\":\"000074bbfb84e7e366ea8f127df3650701f985a15a22519e31fba304d1c7c87a\"}","chainSize":3,"difficulty":4,"totalDifficulty":10,"hashesPerSecond":666666666,"totalExpectedHashes":131328.0,"nonce":16100,"startTime":1730392877619,"endTime":1730392877619}

Number of Blocks on Chain: 3

Client disconnected

**Output: Client 2**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=63166:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task2/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.SigningClientTCP

Public key of client (e, n): (65537, 3496144616041649216529399710387178144417291727192155336399546745064186044334538210845659876876866888308301671884689309909843466279444875361331139659544050516092235476826353760206073492025810368577355208825014597092317715125346745408337620393)

Private key of client (d, n): (318903100762881715922498000651457206575317300840055306178135868928905872606800271975149224773332777794330338503847791241866390796164459840547167320875241231608030954106135777245448672704304364145152921575409518033961231098467883478265346305, 3496144616041649216529399710387178144417291727192155336399546745064186044334538210845659876876866888308301671884689309909843466279444875361331139659544050516092235476826353760206073492025810368577355208825014597092317715125346745408337620393)

Client id: 5601b717faf1cdac68c1cde4f3ef7020fca3a4fc5bbe02f7d31f

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Approximate hashes per second on this machine: 285714285

Expected total hashes required for the whole chain: 256.0

Nonce for most recent block: 157

Chain hash: 00669fde6127f258650b41e1edb549a54ad34fc44abcb8e855d01c89aec303c8

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Bob pays Tony 30 DSCoin

Total execution time to add this block was 260 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 12:43:55.26","Tx ": "Genesis","PrevHash" : "0","nonce" : 157,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 12:44:29.167","Tx ": "Bob pays Tony 30 DSCoin","PrevHash" : "00669fde6127f258650b41e1edb549a54ad34fc44abcb8e855d01c89aec303c8","nonce" : 113396,"difficulty": 4}], "chainHash":"0000c3db87b2141b88139f896f1df3b4c0fd7dedeebb4774e3cf42e19a7ccce9"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 6

Exiting...

Process finished with exit code 0

**Output: Server 2**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=63163:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task2/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.VerifyingServerTCP

Blockchain Server Running

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"getStatus","difficultyOrIndex":0},"digitalSignature":"1266469122514597604586247573315546614525676415267085745834239535219502213964137477565851113310766998681901423431815198228205969574981224462607764702418388375624150704145440362574838139308633967337083983241033359975539927405494184257935913959","publicKeyExponent":"65537","publicKeyModulus":"3496144616041649216529399710387178144417291727192155336399546745064186044334538210845659876876866888308301671884689309909843466279444875361331139659544050516092235476826353760206073492025810368577355208825014597092317715125346745408337620393","clientIdentifier":"5601b717faf1cdac68c1cde4f3ef7020fca3a4fc5bbe02f7d31f"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing status","chainHash":"00669fde6127f258650b41e1edb549a54ad34fc44abcb8e855d01c89aec303c8","chainSize":1,"difficulty":2,"totalDifficulty":2,"hashesPerSecond":285714285,"totalExpectedHashes":256.0,"nonce":157,"startTime":1730393041946,"endTime":1730393041946}

Number of Blocks on Chain: 1

Signature verified successfully.

Total execution time to add this block was 260 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"addBlock","transactionData":"Bob pays Tony 30 DSCoin","difficultyOrIndex":4},"digitalSignature":"2487131857356693249608915875331220889519155722327527372851379516545620047726670847763073164419099181307965886314277796056918098951188519997145028232317163599542848447198850990475997937953105918755870289475318325266140906449723961524948725777","publicKeyExponent":"65537","publicKeyModulus":"3496144616041649216529399710387178144417291727192155336399546745064186044334538210845659876876866888308301671884689309909843466279444875361331139659544050516092235476826353760206073492025810368577355208825014597092317715125346745408337620393","clientIdentifier":"5601b717faf1cdac68c1cde4f3ef7020fca3a4fc5bbe02f7d31f"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"0000c3db87b2141b88139f896f1df3b4c0fd7dedeebb4774e3cf42e19a7ccce9","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":285714285,"totalExpectedHashes":65792.0,"nonce":113396,"startTime":1730393069167,"endTime":1730393069427}

Number of Blocks on Chain: 2

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"viewChain","difficultyOrIndex":0},"digitalSignature":"1351095786871108766764892267275491176023831719168836473178052555753271337979807577304819812561341363863083780449726191960975757033069671226870803166583225409249853744961461486706249730892845878799297197760777379116630183125369448862805109320","publicKeyExponent":"65537","publicKeyModulus":"3496144616041649216529399710387178144417291727192155336399546745064186044334538210845659876876866888308301671884689309909843466279444875361331139659544050516092235476826353760206073492025810368577355208825014597092317715125346745408337620393","clientIdentifier":"5601b717faf1cdac68c1cde4f3ef7020fca3a4fc5bbe02f7d31f"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing blockchain","chainHash":"{\"ds\_chain\" : [{\"index\" : 0,\"time stamp \" : \"2024-10-31 12:43:55.26\",\"Tx \": \"Genesis\",\"PrevHash\" : \"0\",\"nonce\" : 157,\"difficulty\": 2},{\"index\" : 1,\"time stamp \" : \"2024-10-31 12:44:29.167\",\"Tx \": \"Bob pays Tony 30 DSCoin\",\"PrevHash\" : \"00669fde6127f258650b41e1edb549a54ad34fc44abcb8e855d01c89aec303c8\",\"nonce\" : 113396,\"difficulty\": 4}], \"chainHash\":\"0000c3db87b2141b88139f896f1df3b4c0fd7dedeebb4774e3cf42e19a7ccce9\"}","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":285714285,"totalExpectedHashes":65792.0,"nonce":113396,"startTime":1730393072740,"endTime":1730393072740}

Number of Blocks on Chain: 2

Client disconnected

**Output: Client 3**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=63227:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task2/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.SigningClientTCP

Public key of client (e, n): (65537, 1894550011569586982835876890593016328317786428980963016938565880181391488509650874708484201344586972967490695088517293104272988990531665807791035117795572407841818155819895792160417801433467368977338430410307264744964716320873299840731454713)

Private key of client (d, n): (994496383691913444062435521769091470875818068111190468112982642019015670349710993663446198249179563361576130925052594981131148050234376354570346766139265261554365887679752857291206862767176644369421086357690329836106673465878942540765131521, 1894550011569586982835876890593016328317786428980963016938565880181391488509650874708484201344586972967490695088517293104272988990531665807791035117795572407841818155819895792160417801433467368977338430410307264744964716320873299840731454713)

Client id: 7d77238ea3e9b60335bfb500c1e30dbceca2399a4d934c750224

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Approximate hashes per second on this machine: 666666666

Expected total hashes required for the whole chain: 256.0

Nonce for most recent block: 30

Chain hash: 00669396c6eda11e8e6c4c650ce0f17f1a51d706cd2020c2ddf3fbc38ae8e8ec

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 1

Enter difficulty > 1: 4

Enter transaction: Carol pays Donna 10 DSCoin

Total execution time to add this block was 241 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 3

View the Blockchain

{"ds\_chain" : [{"index" : 0,"time stamp " : "2024-10-31 12:45:42.222","Tx ": "Genesis","PrevHash" : "0","nonce" : 30,"difficulty": 2},{"index" : 1,"time stamp " : "2024-10-31 12:46:16.683","Tx ": "Carol pays Donna 10 DSCoin","PrevHash" : "00669396c6eda11e8e6c4c650ce0f17f1a51d706cd2020c2ddf3fbc38ae8e8ec","nonce" : 83811,"difficulty": 4}], "chainHash":"000067551cfad083fe3f66ec84f4db1226e8963d067d0d2a4db0e47dbd11d80c"}

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 2

Chain verification: TRUE

Total execution time required to verify the chain was 0 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

Enter your choice: 6

Exiting...

Process finished with exit code 0

**Output: Server 3**

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=63224:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/khushi.bhuwaniagmail.com/Desktop/Distributed Systems/Project3/Project3Task2/target/classes:/Users/khushi.bhuwaniagmail.com/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.VerifyingServerTCP

Blockchain Server Running

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"getStatus","difficultyOrIndex":0},"digitalSignature":"610627415260011954644815289271922354694025147342121199689363422761738653070372520451268162387295342065451577731955788615933714864582462335520848362021478551770384978736971247584090731987948659992058363934729064272821286692462050266817305487","publicKeyExponent":"65537","publicKeyModulus":"1894550011569586982835876890593016328317786428980963016938565880181391488509650874708484201344586972967490695088517293104272988990531665807791035117795572407841818155819895792160417801433467368977338430410307264744964716320873299840731454713","clientIdentifier":"7d77238ea3e9b60335bfb500c1e30dbceca2399a4d934c750224"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing status","chainHash":"00669396c6eda11e8e6c4c650ce0f17f1a51d706cd2020c2ddf3fbc38ae8e8ec","chainSize":1,"difficulty":2,"totalDifficulty":2,"hashesPerSecond":666666666,"totalExpectedHashes":256.0,"nonce":30,"startTime":1730393150060,"endTime":1730393150060}

Number of Blocks on Chain: 1

Signature verified successfully.

Total execution time to add this block was 241 milliseconds

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"addBlock","transactionData":"Carol pays Donna 10 DSCoin","difficultyOrIndex":4},"digitalSignature":"1599271654976022669542894308178700116118966432530370628105057007829446871989178589141928365522151636814716908472730467344603242025247227368635275793853844181267306320962883301733794566532813583544475790891822451047813378868605090178653667256","publicKeyExponent":"65537","publicKeyModulus":"1894550011569586982835876890593016328317786428980963016938565880181391488509650874708484201344586972967490695088517293104272988990531665807791035117795572407841818155819895792160417801433467368977338430410307264744964716320873299840731454713","clientIdentifier":"7d77238ea3e9b60335bfb500c1e30dbceca2399a4d934c750224"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Block added","chainHash":"000067551cfad083fe3f66ec84f4db1226e8963d067d0d2a4db0e47dbd11d80c","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":666666666,"totalExpectedHashes":65792.0,"nonce":83811,"startTime":1730393176683,"endTime":1730393176924}

Number of Blocks on Chain: 2

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"viewChain","difficultyOrIndex":0},"digitalSignature":"420578613279824900987633950735142956363098668011069606554077002443591802825165765719219689133668957690716051012074376794543139432871572822936025947021441674126300307851926082118384526506325790137910476678438140152633754637635503736363604113","publicKeyExponent":"65537","publicKeyModulus":"1894550011569586982835876890593016328317786428980963016938565880181391488509650874708484201344586972967490695088517293104272988990531665807791035117795572407841818155819895792160417801433467368977338430410307264744964716320873299840731454713","clientIdentifier":"7d77238ea3e9b60335bfb500c1e30dbceca2399a4d934c750224"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"Viewing blockchain","chainHash":"{\"ds\_chain\" : [{\"index\" : 0,\"time stamp \" : \"2024-10-31 12:45:42.222\",\"Tx \": \"Genesis\",\"PrevHash\" : \"0\",\"nonce\" : 30,\"difficulty\": 2},{\"index\" : 1,\"time stamp \" : \"2024-10-31 12:46:16.683\",\"Tx \": \"Carol pays Donna 10 DSCoin\",\"PrevHash\" : \"00669396c6eda11e8e6c4c650ce0f17f1a51d706cd2020c2ddf3fbc38ae8e8ec\",\"nonce\" : 83811,\"difficulty\": 4}], \"chainHash\":\"000067551cfad083fe3f66ec84f4db1226e8963d067d0d2a4db0e47dbd11d80c\"}","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":666666666,"totalExpectedHashes":65792.0,"nonce":83811,"startTime":1730393200910,"endTime":1730393200910}

Number of Blocks on Chain: 2

Signature verified successfully.

We have a visitor

THE JSON REQUEST MESSAGE IS: {"transactionRequest":{"action":"verifyChain","difficultyOrIndex":0},"digitalSignature":"795688769671177786486316079582980776712631435776462790105755090124625078173042614321055244317945650765617761755275344376951064820613758042718897610389004833591343332116199757947821159348844544748712897253548995173195365007565151230917862128","publicKeyExponent":"65537","publicKeyModulus":"1894550011569586982835876890593016328317786428980963016938565880181391488509650874708484201344586972967490695088517293104272988990531665807791035117795572407841818155819895792160417801433467368977338430410307264744964716320873299840731454713","clientIdentifier":"7d77238ea3e9b60335bfb500c1e30dbceca2399a4d934c750224"}

THE JSON RESPONSE MESSAGE IS: {"status":"success","message":"TRUE","chainHash":"000067551cfad083fe3f66ec84f4db1226e8963d067d0d2a4db0e47dbd11d80c","chainSize":2,"difficulty":4,"totalDifficulty":6,"hashesPerSecond":666666666,"totalExpectedHashes":65792.0,"nonce":83811,"startTime":1730393202212,"endTime":1730393202212}

Number of Blocks on Chain: 2

Client disconnected