МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра комп'ютерної інженерії та інформаційних систем

ЛАБОРАТОРНА РОБОТА №8

з дисципліни «Теорія і технології проектування спеціалізованих операційних систем»

ЛРКІ.180102.22.01.02 ПЗ

Галузь знань \_\_\_\_\_\_\_\_\_\_12 – Інформаційні технології\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Спеціальність \_\_\_\_\_\_\_\_\_123 –Комп’ютерна інженерія\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Виконав:

студент 1 курсу, групи КІ2м-22-1 Біньковський Я.В.

(Підпис)

Перевірив: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лисенко С.М.

(Підпис)

Хмельницький – 2023 р

Тема:

Реалізація блокчейн технологій. Побудова блокчейну для нефінансових операцій.

Завдання:

Простий рівень.

Реалізувати приклад, наведений в метод. вказівках. Оформити звіт з виконаної роботи.

Реалізація:

Кроки реалізації:

1. Реалізація майнінгу.
2. Гаманець.
3. Тестування.
4. Входи та виходи
5. Обробка транзакції.

Реалізація майнінгу. Реалізація майнінгу включає в себе перевірку роботи шляхом випробування різних значень змінних у блоці, поки хеш не почне з певної кількості 0. Для цього додано змінну "nonce" і методи "calculateHash()" і "mineBlock()". Майнери починають ітерацію з випадкової точки і можуть спробувати різні числа. Складність визначається як кількість 0, які потрібно знайти. Метод "mineBlock()" виконує майнінг блоку, а метод "isChainValid()" перевіряє правильність хешів блоків. Кожен блок має вирішений хеш, а складність зберігається як статична змінна. Також використовується бібліотека Bouncy Castle для криптографії.

Гаманець. Гаманець - це засіб для зберігання адрес, які використовуються для передачі власності монет в криптовалютах через блокчейн. Гаманці можуть просто зберігати адреси або мати додатковий функціонал для здійснення транзакцій на блокчейні. Створюємо клас Wallet для зберігання відкритого та закритого ключів. Відкритий ключ використовується як адреса в криптовалюті, яку можна поділитися з іншими користувачами для отримання платежів.

Приватний ключ використовується для підпису транзакцій і забезпечення безпеки власності монет.

Клас Wallet має метод generateKeyPair(), який генерує відповідну пару ключів за допомогою криптографічних алгоритмів.

Тестування. На етапі тестування ми вносимо зміни до класу NoobChain. Додаємо нові змінні та замінюємо вміст методу main. Здійснюємо такі кроки:

1. Імпортуємо необхідні бібліотеки та класи.
2. Ініціалізуємо змінні: blockchain - список блоків, difficulty - складність генерації блоків, walletA і walletB - гаманці.
3. Встановлюємо Bouncey Castle як постачальника безпеки.
4. Створюємо нові гаманці (walletA і walletB).
5. Виводимо на екран приватний і публічний ключі walletA.
6. Створюємо тестову транзакцію з walletA до walletB.
7. Підписуємо транзакцію за допомогою приватного ключа walletA.
8. Перевіряємо правильність підпису за допомогою публічного ключа.

Це дозволяє перевірити працездатність деяких важливих аспектів. Після цього залишається лише створити/перевірити вхідні та вихідні дані та зберегти транзакцію в Blockchain.

Входи та виходи. В вхідних та вихідних данних відображається перехід кількості біткоїнів між сторонами. Не додається і не віднімається біткоїн, але вказується, що відправник раніше отримав біткоїн і створено транзакцію, яка показує перехід 1 біткоїна на адресу отримувача. Баланс гаманця визначається сумою невитрачених результатів транзакцій, які адресовані отримувачеві. Використовується термін "невитрачені виходи транзакцій" (UTXO) для визначення таких вхідних даних транзакцій.

Обробка транзакції. Обробка транзакції. Блоки в ланцюжку можуть містити багато транзакцій, і блокчейн може бути дуже довгим. Для обробки нової транзакції може знадобитися значний час, оскільки потрібно знайти та перевірити її вхідні дані. Для уникнення цього, використовується додаткова колекція невитрачених транзакцій, які можна використовувати як вхідні дані. У класі NoobChain ця колекція всіх UTXO додається як частина функціоналу. Клас NoobChain має ArrayList блоків, HashMap UTXOs, змінні difficulty, walletA та walletB. Використовуючи HashMaps, ми можемо шукати значення за ключем.

Висновок:

Гаманці тепер мають можливість надійно надсилати кошти на блокчейн, лише якщо в них є достатньо коштів для надсилання. Це означає, що є власна локальна криптовалюта.

Блокчейн вміє:

* дозволяти користувачам створювати гаманці за допомогою 'new Wallet()';
* надавати гаманці з відкритими та приватними ключами за допомогою криптографії з еліптичною кривою;
* забезпечувати переказ коштів за допомогою алгоритму цифрового підпису для підтвердження права власності;
* дозволяти користувачам здійснювати транзакції у блокчейні за допомогою "Block.addTransaction(walletA.sendFunds(walletB.publicKey, 20))".

Код:

public class NoobChain {

public static ArrayList<Block> blockchain = new ArrayList<Block>();

public static HashMap<String,TransactionOutput> UTXOs = new

HashMap<String,TransactionOutput>();

public static int difficulty = 3;

public static float minimumTransaction = 0.1f;

public static Wallet walletA;

public static Wallet walletB;

public static Transaction genesisTransaction;

public static void main(String[] args) {

//add our blocks to the blockchain ArrayList:

Security.addProvider(new

org.bouncycastle.jce.provider.BouncyCastleProvider()); //Setup Bouncey castle as

a Security Provider

//Create wallets:

walletA = new Wallet();

walletB = new Wallet();

Wallet coinbase = new Wallet();

//create genesis transaction, which sends 100 NoobCoin to

walletA:

genesisTransaction = new Transaction(coinbase.publicKey,

walletA.publicKey, 100f, null);

genesisTransaction.generateSignature(coinbase.privateKey);

//manually sign the genesis transaction

genesisTransaction.transactionId = "0"; //manually set the

transaction id

genesisTransaction.outputs.add(new

TransactionOutput(genesisTransaction.reciepient, genesisTransaction.value,

genesisTransaction.transactionId)); //manually add the Transactions Output

UTXOs.put(genesisTransaction.outputs.get(0).id,

genesisTransaction.outputs.get(0)); //its important to store our first

transaction in the UTXOs list.

System.out.println("Creating and Mining Genesis block... ");

Block genesis = new Block("0");

genesis.addTransaction(genesisTransaction);

addBlock(genesis);

//testing

Block block1 = new Block(genesis.hash);

System.out.println("\nWalletA's balance is: " +

walletA.getBalance());

System.out.println("\nWalletA is Attempting to send funds (40) to

WalletB...");

block1.addTransaction(walletA.sendFunds(walletB.publicKey, 40f));

addBlock(block1);

System.out.println("\nWalletA's balance is: " +

walletA.getBalance());

System.out.println("WalletB's balance is: " +

walletB.getBalance());

Block block2 = new Block(block1.hash);

System.out.println("\nWalletA Attempting to send more funds

(1000) than it has...");

block2.addTransaction(walletA.sendFunds(walletB.publicKey,

1000f));

addBlock(block2);

System.out.println("\nWalletA's balance is: " +

walletA.getBalance());

System.out.println("WalletB's balance is: " +

walletB.getBalance());

Block block3 = new Block(block2.hash);

System.out.println("\nWalletB is Attempting to send funds (20) to

WalletA...");

block3.addTransaction(walletB.sendFunds( walletA.publicKey, 20));

System.out.println("\nWalletA's balance is: " +

walletA.getBalance());

System.out.println("WalletB's balance is: " +

walletB.getBalance());

isChainValid();

}

public static Boolean isChainValid() {

Block currentBlock;

Block previousBlock;

String hashTarget = new String(new

char[difficulty]).replace('\0', '0');

HashMap<String,TransactionOutput> tempUTXOs = new

HashMap<String,TransactionOutput>(); //a temporary working list of unspent

transactions at a given block state.

tempUTXOs.put(genesisTransaction.outputs.get(0).id,

genesisTransaction.outputs.get(0));

//loop through blockchain to check hashes:

for(int i=1; i < blockchain.size(); i++) {

currentBlock = blockchain.get(i);

previousBlock = blockchain.get(i-1);

//compare registered hash and calculated hash:

if(!currentBlock.hash.equals(currentBlock.calculateHash()) ){

System.out.println("#Current Hashes not equal");

return false;

}

//compare previous hash and registered previous hash

if(!previousBlock.hash.equals(currentBlock.previousHash)

) {

System.out.println("#Previous Hashes not equal");

return false;

}

//check if hash is solved

if(!currentBlock.hash.substring( 0,

difficulty).equals(hashTarget)) {

System.out.println("#This block hasn't been

mined");

return false;

}

//loop thru blockchains transactions:

TransactionOutput tempOutput;

for(int t=0; t <currentBlock.transactions.size(); t++) {

Transaction currentTransaction =

currentBlock.transactions.get(t);

if(!currentTransaction.verifiySignature()) {

System.out.println("#Signature on

Transaction(" + t + ") is Invalid");

return false;

}

if(currentTransaction.getInputsValue() !=

currentTransaction.getOutputsValue()) {

System.out.println("#Inputs are note equal

to outputs on Transaction(" + t + ")");

return false;

}

for(TransactionInput input:

currentTransaction.inputs) {

tempOutput =

tempUTXOs.get(input.transactionOutputId);

if(tempOutput == null) {

System.out.println("#Referenced

input on Transaction(" + t + ") is Missing");

return false;

}

if(input.UTXO.value != tempOutput.value) {

System.out.println("#Referenced

input Transaction(" + t + ") value is Invalid");

return false;

}

tempUTXOs.remove(input.transactionOutputId);

}

for(TransactionOutput output:

currentTransaction.outputs) {

tempUTXOs.put(output.id, output);

}

if( currentTransaction.outputs.get(0).reciepient

!= currentTransaction.reciepient) {

System.out.println("#Transaction(" + t +

") output reciepient is not who it should be");

return false;

}

if( currentTransaction.outputs.get(1).reciepient

!= currentTransaction.sender) {

System.out.println("#Transaction(" + t +

") output 'change' is not sender.");

return false;

}

}

}

System.out.println("Blockchain is valid");

return true;

}

public static void addBlock(Block newBlock) {

newBlock.mineBlock(difficulty);

blockchain.add(newBlock);

}

}