УДК 004

Создание транзакций блокчейн с использованием Java

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье будут рассмотрены возможности создания транзакций блокчейн. Так же будет внедрена система проверки доказательства работы. Данная работа будет выполнена в среде разработки IntelijIdea с использованием языка программирования Java.
Ключевые слова: Java, Blockchein, Bitcoin

Creating blockchain transactions using Java

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article will consider the possibilities of creating transaction blockchain. A proof-of-work verification system will also be introduced. This work will be done in the IntelijIdea development environment using the Java programming language.
Keywords: Java, Blockchein, Bitcoin

В криптовалютах право собственности на монеты передается в блокчейне в виде транзакций, участники имеют адрес, на который и с которого могут быть отправлены средства. В своей базовой форме кошельки могут просто хранить эти адреса, однако большинство кошельков также представляют собой программное обеспечение, способное совершать новые транзакции в блокчейне.

Цель работы – создать транзакции в технологии блокчейн, так же создать кошельки с которого и на который будут поступать средства.

Исследованиями в данной теме занимались следующие авторы.

Б. Кумалаков и Я. Шакан предложили разработку децентрализованного студенческого приложения, который будет обрабатывать и хранить токены, которые будут представлять собой кредиты, которые студенты будут получать за прохождение определенных курсов [1]. В.П.Часовских, В.Г. Лабунец, М.П. Воронов описали технологию блокчейн как основу развития и повышения эффективности электронной системы ВУЗа [2]. С.Г. Наумов в своей работе представил технологию блокчейн, способы добытия криптовалюты [3]. П.А. Андреева, Р.А. Андреев провели обзор на разновидности технологий блокчейн и видах их применения [4]. Л.А. Симанович, расписала возможности использования блокчейн для электронных торговых площадок [5].

В начале создадим класс Wallet для хранения открытого и закрытого ключей (рис.1).

```
import java.security.*;

public class Wallet {
        public PrivateKey privateKey;
        public PublicKey publicKey;
}
```

Рисунок 1 – создание класса Wallet

Для монеты открытый ключ будет действовать как адрес. Можно поделиться этим открытым ключом с другими, чтобы получить платеж. Закрытый ключ используется для подписи транзакций, поэтому никто не может потратить монеты, кроме владельца закрытого ключа. Закрыты ключ необходимо держать в тайне. Открытый ключ отправляется так же вместе с транзакцией, и его можно использовать для проверки правильности подписи и отсутствия подделки данных.

Генерируем закрытые и открытые ключи в «KeyPair». Будем использовать криптографию с эллиптическими кривыми для генерации пар ключей. Добавим метод «agenerateKeyPair()» в класс «Wallet» и вызовем его в конструкторе (рис.2).

```
import java.security.*;

public class Wallet {

        public PrivateKey privateKey;
        public PublicKey publicKey;

        public Wallet(){
                generateKeyPair();
        }

        public void generateKeyPair() {
                try {
                        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("ECDSA","BC");
                        SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
                        ECGenParameterSpec ecSpec = new ECGenParameterSpec("prime192v1");
                        keyGen.initialize(ecSpec, random);
                        KeyPair keyPair = keyGen.generateKeyPair();
                        privateKey = keyPair.getPrivate();
                        publicKey = keyPair.getPublic();
                }catch(Exception e) {
                        throw new RuntimeException(e);
                }
        }

}
```

Рисунок 2 – генерация пар ключей

Теперь, когда есть схема класса кошелька, можно приступать к реализации транзакций.

Каждая транзакция будет нести определенный объем данных:

- Публичный ключ отправителя средств.
- Публичный ключ получателя средств.
- Сумма переводимых средств.
- Входные данные, которые являются ссылками на предыдущие транзакции, которые доказывают, что у отправителя есть средства для отправки.
- Выходные данные, которые показывают количество релевантных адресов, полученных в транзакции.
- Криптографическая подпись, которая доказывает, что владелец адреса является тем, кто отправляет эту транзакцию, и что данные не были изменены. (предотвращение изменения отправленной суммы третьей стороной)

Теперь создадим новый класс Transaction (рис.3).

```
public class Transaction {

    public String transactionId;
    public PublicKey sender;
    public PublicKey reciepient;
    public float value;
    public byte[] signature;

    public ArrayList<TransactionInput> inputs = new ArrayList<TransactionInput>();
    public ArrayList<TransactionOutput> outputs = new ArrayList<TransactionOutput>();

    private static int sequence = 0;

    public Transaction(PublicKey from, PublicKey to, float value,  ArrayList<TransactionInput> inputs) {
        this.sender = from;
        this.reciepient = to;
        this.value = value;
        this.inputs = inputs;
    }
    private String calulateHash() {
        sequence++;
        return StringUtil.applySha256(
                input: StringUtil.getStringFromKey(sender) +
                        StringUtil.getStringFromKey(reciepient) +
                        Float.toString(value) + sequence
        );
    }
```

Рисунок 3 – Создание класса Transaction

Необходимо создать пустые классы «TransactionInput» и «TransactionOutput», они будут заполнены позже.

Класс транзакций также будет содержать соответствующие методы для генерации и проверки подписи.

Подписи выполняют две очень важные задачи в цепочке блоков: во-первых, они позволяют только владельцу тратить свои монеты , во-вторых, они не позволяют другим вмешиваться в их отправленную транзакцию до того, как будет добыт новый блок.

Закрытый ключ используется для подписи данных, а открытый ключ может использоваться для проверки их целостности.

Как видно из предыдущего блока кода, что подпись будет набором байтов, поэтому создадим метод для их генерации. Первое, что понадобится, это несколько вспомогательных функций в классе StringUtil (рис.4).

```
        public static byte[] applyECDSASig(PrivateKey privateKey, String input) {
        Signature dsa;
        byte[] output = new byte[0];
        try {
                dsa = Signature.getInstance("ECDSA", "BC");
                dsa.initSign(privateKey);
                byte[] strByte = input.getBytes();
                dsa.update(strByte);
                byte[] realSig = dsa.sign();
                output = realSig;
        } catch (Exception e) {
                throw new RuntimeException(e);
        }
        return output;
    }


    public static boolean verifyECDSASig(PublicKey publicKey, String data, byte[] signature) {
        try {
                Signature ecdsaVerify = Signature.getInstance("ECDSA", "BC");
                ecdsaVerify.initVerify(publicKey);
                ecdsaVerify.update(data.getBytes());
                return ecdsaVerify.verify(signature);
        }catch(Exception e) {
                throw new RuntimeException(e);
        }
    }

    public static String getStringFromKey(Key key) {
        return Base64.getEncoder().encodeToString(key.getEncoded());
    }
```

Рисунок 4 – Создание класса StringUtil

Теперь применим эти методы подписи в классе «Transaction», добавив методы «generateSignature()» и «verifiySignature()» (рис.5).

```
public void generateSignature(PrivateKey privateKey) {
        String data = StringUtil.getStringFromKey(sender) + StringUtil.getStringFromKey(reciepient) + Float.toString(value)     ;
        signature = StringUtil.applyECDSASig(privateKey,data);
}

public boolean verifiySignature() {
        String data = StringUtil.getStringFromKey(sender) + StringUtil.getStringFromKey(reciepient) + Float.toString(value)     ;
        return StringUtil.verifyECDSASig(sender, data, signature);
}
```

Рисунок 5 – Добавление методов в класс Transaction

Подписи будут проверяться майнерами по мере добавления новой транзакции в блок.

Теперь проверим, как это работает. В классе запуска добавим несколько новых переменных и заменим содержимое основного метода (рис.6).

```
public class MyChain {

        public static ArrayList<Block> blockchain = new ArrayList<Block>();
        public static int difficulty = 5;
        public static Wallet walletA;
        public static Wallet walletB;

        public static void main(String[] args) {

                Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());

                walletA = new Wallet();
                walletB = new Wallet();

                System.out.println("Приватный и публичный ключ:");
                System.out.println(StringUtil.getStringFromKey(walletA.privateKey));
                System.out.println(StringUtil.getStringFromKey(walletA.publicKey));

                Transaction transaction = new Transaction(walletA.publicKey, walletB.publicKey, 5, null);
                transaction.generateSignature(walletA.privateKey);

                System.out.println("Подпись подтверждена");
                System.out.println(transaction.verifiySignature());

        }
```

Рисунок 6 – Добавление значений для проверки

Будет создано два кошелька, «walletA» и «walletB», затем выводится закрытый и открытый ключи «walletA». Создаем транзакцию и подписываем ее с помощью закрытого ключа «walletA». После запуска вывод будет следующим (рис.7).

```
Приватный и публий ключи:
MDkCAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQEEHzAdAgEBBBgKDrAR9B5lLh2QyxX48MGe5SpzraZrJlk=
MEkwEwYHKoZIzj0CAQYIKoZIzj0DAQEDMgAEbwxxE8hhqwTmlMyTEindql16zcgTfZSHk3i0nFVtTHcjBEDVk4BkBiyKQK+QYcFo
Подпись подтверждена
true
```

Рисунок 7 – Пример работы создания ключей и проверки

Чтобы иметь 1 биткойн, необходимо получить 1 биткойн. Программное обеспечение криптовалюты на самом деле не добавляет один биткойн получателю и забирает один биткойн от отправителя, отправитель сослался на то, что он ранее получил один биткойн, затем был создан вывод транзакции, показывающий, что 1 биткойн был отправлен на другой адрес. Входные данные транзакции являются ссылками на предыдущие выходные данные.

Баланс кошелька — это сумма всех неизрасходованных выходов транзакций. Неизрасходованные выходы транзакций называются – UTXO.

Теперь создадим класс «TransactionInput» (рис.8).

```
public class TransactionInput {
    public String transactionOutputId;
    public TransactionOutput UTXO;

    public TransactionInput(String transactionOutputId) {
        this.transactionOutputId = transactionOutputId;
    }
}
```

Рисунок 8 – Создание класса TransactionInput

Этот класс будет использоваться для ссылки на «TransactionOutputs», которые еще не были потрачены. «TransactionOutputId» будет использоваться для поиска, соответствующего «TransactionOutput», что позволит майнерам проверить право собственности.

Создадим класс «TransactionOutput» (рис.9).

```
public class TransactionOutput {
    public String id;
    public PublicKey reciepient;
    public float value;
    public String parentTransactionId;


    public TransactionOutput(PublicKey reciepient, float value, String parentTransactionId) {
        this.reciepient = reciepient;
        this.value = value;
        this.parentTransactionId = parentTransactionId;
        this.id = StringUtil.applySha256( input: StringUtil.getStringFromKey(reciepient)+Float.toString(value)+parentTransactionId);
    }


    public boolean isMine(PublicKey publicKey) {
        return (publicKey == reciepient);
    }

}
```

Рисунок 9 – Создание Класса TransactionOutput.

Выходы транзакции покажут окончательную сумму, отправленную каждой стороне транзакции. Когда они упоминаются как входы в новых транзакциях, они служат доказательством того, что у отправителя есть монеты для отправки.

Блоки в цепочке могут получать множество транзакций, а блокчейн может быть очень длинным, обработка новой транзакции может занять много времени, потому что нужно найти и проверить ее входные данные. Чтобы обойти это, будем хранить дополнительную коллекцию всех неизрасходованных транзакций, которые можно использовать в качестве входных данных. В класс MyCoin добавим эту коллекцию всех UTXO (рис.10).

```
    public static ArrayList<Block> blockchain = new ArrayList<Block>();
    public static HashMap<String,TransactionOutputs> UTXOs = new HashMap<String,TransactionOutputs>();
    public static int difficulty = 5;
    public static Wallet walletA;
    public static Wallet walletB;
```

Рисунок 10 – Добавление коллекции UTXO

Теперь соберем все вместе для обработки транзакции с помощью логического метода «processTransaction» в классе транзакций (рис.11).

```
public boolean processTransaction() {
    if(verifySignature() == false) {
        System.out.println("#Не удалось проверить подпись транзакции");
        return false;
    }
    for(TransactionInput i : inputs) {
        i.UTXO = MyChain.UTXOs.get(i.transactionOutputId);
    }
    if(getInputsValue() < MyChain.minimumTransaction) {
        System.out.println("Транзакционные входы слишком малы: " + getInputsValue());
        System.out.println("Пожалуйста, введите сумму больше, чем " + MyChain.minimumTransaction);
        return false;
    }
    float leftOver = getInputsValue() - value;
    transactionId = calulateHash();
    outputs.add(new TransactionOutput( this.reciepient, value,transactionId));
    outputs.add(new TransactionOutput( this.sender, leftOver,transactionId));
    for(TransactionOutput o : outputs) {
        MyChain.UTXOs.put(o.id , o);
    }
    for(TransactionInput i : inputs) {
        if(i.UTXO == null) continue;
        MyChain.UTXOs.remove(i.UTXO.id);
    }
    return true;
}
```

Рисунок 11 – Создание метода

С помощью этого метода будут выполняться некоторые проверки, чтобы убедиться, что транзакция действительна, затем собираем входные данные и генерируем выходные данные.

Важно отметить, что ближе к концу отбрасываются входы из списка UTXO, это означает, что выход транзакции может использоваться только один раз в качестве входа.

Готова работающая система транзакций, теперь нужно внедрить ее в цепочку блоков. Для этого необходимо заменить бесполезные данные, которые были в блоках, на ArrayList. Однако в одном блоке может быть 1000 транзакций, слишком много, чтобы включить их в расчет хэш, поэтому будем использовать корень меркла транзакций.

Добавим вспомогательный метод для создания «merkleroot» в «StringUtils» (рис.12).

```
public static String getMerkleRoot(ArrayList<Transaction> transactions) {
    int count = transactions.size();

    List<String> previousTreeLayer = new ArrayList<String>();
    for(Transaction transaction : transactions) {
        previousTreeLayer.add(transaction.transactionId);
    }
    List<String> treeLayer = previousTreeLayer;

    while(count > 1) {
        treeLayer = new ArrayList<String>();
        for(int i=1; i < previousTreeLayer.size(); i+=2) {
            treeLayer.add(applySha256( input: previousTreeLayer.get(i-1) + previousTreeLayer.get(i)));
        }
        count = treeLayer.size();
        previousTreeLayer = treeLayer;
    }

    String merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : "";
    return merkleRoot;
}
```

Рисунок 12 – Добавление метода

Теперь можно протестировать отправку монет, а также обновить проверку валидности блокчейна. Но сначала нужен способ ввести в микс новые монеты. Существует множество способов создания новых монет, например, в блокчейне биткойнов: майнеры могут включать транзакцию в себя в качестве вознаграждения за каждый добытый блок. Однако сейчас мы просто выпустим все монеты, которые хотим иметь, в первом блоке. Как и в биткойнах, мы будем жестко кодировать генезис-блок.

Обновим класс MyChain:

- Блок Genesis, который присваивает 100 монет на кошелек A.
- Обновленная проверка валидности цепочки, учитывающая транзакции.
- Несколько тестовых транзакций, чтобы убедиться, что все работает. (рис.13-14).

```
public static void main(String[] args) {
    Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
    walletA = new Wallet();
    walletB = new Wallet();
    Wallet coinbase = new Wallet();
    genesisTransaction = new Transaction(coinbase.publicKey, walletA.publicKey,  value: 100f,  inputs: null);
    genesisTransaction.generateSignature(coinbase.privateKey);
    genesisTransaction.transactionId = "0";
    genesisTransaction.outputs.add(new TransactionOutput(genesisTransaction.reciepient, genesisTransaction.value, genesisTransaction.transactionId));
    UTXOs.put(genesisTransaction.outputs.get(0).id, genesisTransaction.outputs.get(0));
```

Рисунок 13 – Обновление класса запуска

```
System.out.println("Создание и нахождение блока... ");
Block genesis = new Block( previousHash: "0");
genesis.addTransaction(genesisTransaction);
addBlock(genesis);

Block block1 = new Block(genesis.hash);
System.out.println("\nWalletA - баланс равен: " + walletA.getBalance());
System.out.println("\nWalletA отправляет 40 монет WalletB...");
block1.addTransaction(walletA.sendFunds(walletB.publicKey,  value: 40f));
addBlock(block1);
System.out.println("\nWalletA - баланс равен: " + walletA.getBalance());
System.out.println("WalletB - баланс равен: " + walletB.getBalance());

Block block2 = new Block(block1.hash);
System.out.println("\nWalletA отправляет монет (1000) больше, чем имеется на балансе...");
block2.addTransaction(walletA.sendFunds(walletB.publicKey,  value: 1000f));
addBlock(block2);
System.out.println("\nWalletA - баланс равен: " + walletA.getBalance());
System.out.println("WalletB - баланс равен: " + walletB.getBalance());

Block block3 = new Block(block2.hash);
System.out.println("\nWalletB отправляет 20 монет WalletA...");
block3.addTransaction(walletB.sendFunds( walletA.publicKey,  value: 20));
System.out.println("\nWalletA - баланс равен: " + walletA.getBalance());
System.out.println("WalletB - баланс равен: " + walletB.getBalance());

isChainValid();
```

Рисунок 14 – Обновление класса запуска

После запустим программу и проверим, как это работает (рис.15).

```
Создание и нахождение блока...
Транзакция успешно добавлена в блок
Найден блок : 00030c0198510976e9b286fd5c18a0137842dea88adab03709d977dea2cc8563

WalletA - баланс равен: 100.0

WalletA отправляет 40 монет WalletB...
Транзакция успешно добавлена в блок
Найден блок : 000036edc1fb9d78e3d20b5f1dd4179b5adb4849c2334f1ffbb3cd883e1d9316

WalletA - баланс равен: 60.0
WalletB - баланс равен: 40.0

WalletA отправляет монет (1000) больше, чем имеется на балансе...
#Недостаточно средств для отправки транзакции. Транзакция отклонена.
Найден блок : 0009eeeb502a7fc22c29dbb9f28648c37d487f4c849a55c6b647144cd445994e

WalletA - баланс равен: 60.0
WalletB - баланс равен: 40.0

WalletB отправляет 20 монет WalletA...
Транзакция успешно добавлена в блок

WalletA - баланс равен: 80.0
WalletB - баланс равен: 20.0
Блокчейн действителен
```

Рисунок 15 – Вывод программы

Кошельки теперь могут безопасно отправлять средства на блокчейн, только если у них есть средства для отправки. Это означает, что была разработана собственная локальная криптовалюта.

Библиографический список

1. Кумалаков Б. и Шакан Я. Блокчейн в образовании: как управлять студенческим зачетом вуза через блокчейн? // Вестник алматинского университета энергетики и связи 2020. №2(49). С. 128-133.
2. Часовских В.П., Лабунец В.Г., Воронов М.П. Технология "блокчейн" (blockchain) в образовании вузов и цифровой экономике // Эко-потенциал 2018. №2(18). С. 99 - 105.
3. Наумов С.Г. Блокчейн, криптовалюта и майнинг // Евразийская адвокатура 2021. №2(51). С. 32.
4. Андреева П.А., Андреев Р.А. Обзор технологии блокчейн: виды блокчейна и их применение // Интеллектуальные системы в производстве 2019. №1.

С. 11-14.

5. Симанович Л.А. Возможности использования технологии блокчейн (blockchain) для электронной торговой площадки // Национальная безопасность и стратегическое планирование 2019. №1(21). С. 134-140.