Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
Национальный исследовательский университет
”Высшая школа экономики”
Московский институт электроники и математики Национального исследовательского университета ”Высшая школа экономики”
ОП ”Компьютерная безопасность”

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ ”ОРГАНИЗАЦИОННОЕ И ПРАВОВОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ”

ТЕХНОЛОГИЯ «БЛОКЧЕЙН» - СОДЕРЖАНИЕ, ПЕРСПЕКТИВЫ ПРИМЕНЕНИЯ, ПРОБЛЕМЫ БЕЗОПАСНОСТИ. СМАРТ-КОНТРАКТЫ

Автор:
Смирнов Тимофей Богданович,
студент группы СКБ221

Москва, 2024

Оглавление

1 Введение

Кому принадлежат данные, которые мы храним на облачных сервисах? Данные хранятся на центральном сервере и фактически принадлежат владельцу вычислительных мощностей. В одноранговой же сети все узлы равноправны, они хранят копии данных, отключение одного или нескольких из них не приводит к потере данных. Такой подход позволяет пользователям быть владельцами своих данных.

Актуальность: Тема актуальна, так как в современном мире безопасность конфиденциальных данных и криптовалюта имеют растущий интерес.

Проблема: Люди часто становятся жертвами мошенничества из-за кражи личных данных. Проблема недоверительной сети решается использованием криптографии.

Цель: Показать эффективность и целесообразность использования одноранговых распределенных сетей.

Задачи:

- Изучение литературы по теме.
- Проведение сравнительного анализа одноранговой сети и сети клиент/сервер.
- Подведение проблем безоасности технологии блокчейн.
- Разбор работы хеш-функции SHA-256.
- Написание смарт-контракта на языке программирования Solidity.

Гипотеза:

Гипотезой моей работы предполагалось, что применение одноранговых распределенных сетей приведет к увеличению безопасности и эффективности многопользовательских сетей.

Настоящая работа состоит из трез разделов. В первом рассмотрены основные понятия технологии Блокчейн, работа одноранговой сети. Второй раздел посвящен умным контрактам и написан такой смарт-контракт. В третьем разделе рассмотрены проблемы безопасности Блокчейн.

2 Блокчейн - основные понятия

2.1 Распределенные реестры

Распределенные реестры (Distributed Ledger Technology, DLT) - это технологии, которые позволяют хранить и обмениваться данными без необходимости централизованной структуры или управления.

2.2 Блокчейн - определение

Блокчейн представляет собой специфическую форму распределенного реестра, где данные организованы в виде цепочки блоков, каждый из которых содержит информацию о транзакциях или других событиях, а также ссылку на предыдущий блок. Это обеспечивает прозрачность, надежность и безопасность данных, делая блокчейн популярным инструментом для различных приложений, включая криптовалюты, управление цепочками поставок, финансовые услуги и многое другое.

2.3 Блокчейн - история создания

Централизованные системы хранения электронных документов подвержены изменению и подделке, поскольку данные вносятся вручную.

В 1991 году Стюарт Хабер и Скотт Сторнетт разработали прототип блокчейна, который решал эту проблему путем проставления временных меток на электронные документы, что предотвращало их заднее датирование и фальсификацию. Затем документы сортировались по этим меткам и объединялись в блоки.

Блокчейн основан на децентрализованной сети однорангового типа, в которой каждый узел выступает как сервер и клиент одновременно.

Значительный вклад в развитие блокчейна внес Адам Бэк с помощью алгоритма доказательства работы Hashcash. Он добавлял к заголовкам электронных писем текстовые метки, указывающие на затраченные отправителями ресурсы для их вычисления. Это усложнило проведение спам- и DDoS-атак на почтовые серверы.

В 2008 году Сатоши Накамото, личность которого остается неизвестной, опубликовал техническую документацию децентрализованной одноранговой платежной системы Биткойн, навсегда изменив облик Интернета.

2.4 Задача блокчейна

При проектировании первого блокчейн протокола (Биткоин) решалась задача создания одноранговой системы электронных денег позволяющей участникам совершать электронные транзакции без посредников.

2.5 Преимущества одноранговой сети

Данные, которые мы храним на облачных сервисах, находятся на центральных серверах и принадлежат владельцам этих серверов. В одноранговой сети все узлы равны, каждый из них хранит копии данных, и потеря одного или нескольких узлов не приводит к утрате информации. Такой подход позволяет пользователям контролировать свои данные. Все участники равноправны. Клиенты сети одновременно являются клиентами и серверами.

Преимущества:

- Среда пониженного доверия − нет требования к добросовестному поведению участников. То есть сеть остается работоспособной даже в случае некорректной работы участников в случае, если их количество не превышает порогового значения — 50% от всей сети .
- Отказоустойчивость — устранение центрального сервера как главной точки отказа (копии данных хранятся на всех равноправных узлах)
- Предотвращение DDOS (Distributed Deny Of Services) - в отличие от централизованной системы, данные всегда можно получить с другого узла.
- Прозрачность − все данные доступны для публичного просмотра
- Сообщество − все данные, правила функционирования определяются сообществом.

- Неизменяемость − Данные в текущем блоке требуют криптографического подтверждения данных из предыдущего блока. Однажды принятые сетью и сохраненные данные нельзя изменить или подделать.

2.6 Хеш-функции и хеширование информации

Хеш – это результат выполнения хеш-функции, которая принимает некоторый файл или текст (последовательность данных любой длины) и выдает уникальную для входных данных битовую последовательность фиксированной длины. Стоит отметить, что хеш-функция обладает свойством необратимости.

Явление, когда для разных входных данных функция выдает один и тот же результат, называется коллизией. Хеш-функции с коллизиями являются небезопасными и не используются в работе.

2.6.1 Хеш-алгоритм SHA-256

Рассмотрим алгоритм работы хеш-функции SHA-256 на примере входа «miem cs hse».

Этап 1. Обработка входных данных

1. Преведем строку «miem cs hse» в двоичный код:

```
01101101 01101001 01100101 01101101 00100000 01100011
01110011 00100000 01101000 01110011 01100101
```

2. Припишем единицу в конце:

```
01101101 01101001 01100101 01101101 00100000 01100011
01110011 00100000 01101000 01110011 01100101 1
```

3. Дополняем нулями, пока общая длина данных (без учета последних 64 бит) не станет кратной 512 - это 448 бит:

```
01101101 01101001 01100101 01101101 00100000 01100011 01110011
00100000 01101000 01110011 01100101 10000000 00000000 00000000
...
...
00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

4. Припишем еще 64 бита, представляющие число, которое обозначает длину исходных данных в двоичной системе. На примере это число равно 88, что в двоичной системе выглядит как «1011000».

```
01101101 01101001 01100101 01101101 00100000 01100011 01110011
00100000 01101000 01110011 01100101 10000000 00000000 00000000
...
...
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
...
00000000 00000000 00000000 00000000 00000000 00000000 00000000
01011000
```

   В результате получены данные, кратные 512, что поможет нам далее.

Этап 2. Обработка начального значения хеша

Устанавливаем восемь переменных-констант h1-h7, равных первым 32 битам дробных частей квадратных корней первых восьми простых чисел.

```
0x6a09e667
0xbb67ae85
0x3c6ef372
0xa54ff53a
0x510e527f
0x9b05688c
0x1f83d9ab
0x5be0cd19
```

Этап 3. Обработка 64 значений переменных

Каждая константа содержит первые 32 бита дробных частей кубических корней первых 64 простых чисел в диапазоне от 2 до 311.

```
0x428a2f98 0x71374491 0xb5c0fbcf 0xe9b5dba5 0x3956c25b 0x59f111f1
0x923f82a4 0xab1c5ed5 0xd807aa98 0x12835b01 0x243185be 0x550c7dc3
0x72be5d74 0x80deb1fe 0x9bdc06a7 0xc19bf174 0xe49b69c1 0xefbe4786
0x0fc19dc6 0x240ca1cc 0x2de92c6f 0x4a7484aa 0x5cb0a9dc 0x76f988da
0x983e5152 0xa831c66d 0xb00327c8 0xbf597fc7 0xc6e00bf3 0xd5a79147
0x06ca6351 0x14292967 0x27b70a85 0x2e1b2138 0x4d2c6dfc 0x53380d13
0x650a7354 0x766a0abb 0x81c2c92e 0x92722c85 0xa2bfe8a1 0xa81a664b
0xc24b8b70 0xc76c51a3 0xd192e819 0xd6990624 0xf40e3585 0x106aa070
0x19a4c116 0x1e376c08 0x2748774c 0x34b0bcb5 0x391c0cb3 0x4ed8aa4a
0x5b9cca4f 0x682e6ff3 0x748f82ee 0x78a5636f 0x84c87814 0x8cc70208
0x90befffa 0xa4506ceb 0xbef9a3f7 0xc67178f2
```

Этап 4. Главный повторяющийся блок - цикл

Для каждого элемента входных данных будут выполнены следующие действия.

Значения хеш-функций h0–h7 будут изменяться на каждой итерации цикла для получения окончательного результата. Исходное сообщение ”miem cs hse”относите короткое, поэтому будет обрабатываться только один фрагмент.

Этап 5. Формируем последовательность текстов

1. Переносим входные данные с первого этапа в массив, где каждая запись составляет 32-битный элемень. Затем добавляем 48 нулевых элементов, чтобы получить массив [1...64]:

```
01101101011010010110010101101101 00100000011000110111001100100000
01101000011100110110010110000000 00000000000000000000000000000000
00000000000000000000000000000000 00000000000000000000000000000000
...(null elements)
00000000000000000000000000000000 00000000000000000000000000000000
00000000000000000000000000000000 00000000000000000000000000000000
00000000000000000000000000000000 00000000000000000000000001011000
00000000000000000000000000000000 00000000000000000000000000000000
...(null elements)
00000000000000000000000000000000 00000000000000000000000000000000
```

2. Применим код цикла для заполнения пустых элементов:

```
For i from w[16...63]:
   s0 = (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor
      (w[i-15] righthift 3)
   s1 = (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2]
      righthift 10)
   w [i] = w[i-16] + s0 + w[i-7] + s1
```

Результат массива элементов:

```
01101101011010010110010101101101 00100000011000110111001100100000
01101000011100110110010110000000 00000000000000000000000000000000
...(null elements)
00000000000000000000000000000000 00000000000000000000000001011000
11010011101111010001000100001011 00000001000011111001100101111011
...
00110111010001110000001000110111 11000010110000101110101100010110
10000110110100001100000000110001 10100010110101000110011110011010
```

Этап 6. Блок повторений, сжиающий ввод

Начнем, инициализировав переменные a - h, запишем в них ззначения h0 - h7.

После чего изменим значения переменных a...h, запустив цикл сжатия. Этот цикл представляет собой следующее:

```
for i from 0 to 63:
    ch = (e and f) xor ((not e) and g)
    maj = (a and b) xor (a and c) xor (b and c)
    S1 = (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
    S0 = (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
    temp1 = h + S1 + ch + k[i] + w[i]
    temp2 := S0 + maj
    d = c; c = b; b = a; a = temp1 + temp2
    h = g; g = f; f = e; e = d + temp1;
```

Этап 7. Дополнительные правки хешей

После завершения цикла сжатия, но в главном цикле изменяем значения хеша, путем добавления к ним переменных a...h.

```
h0 = h0 + a = f0a4b15f -> 11110000101001001011000101011111
h1 = h1 + b = 3ba0d2f8 -> 00111011101000001101001011111000
h2 = h2 + c = 0c9cfcb6 -> 00001100100111001111110010110110
h3 = h3 + d = 985564e3 -> 10011000010101010110010011100011
h4 = h4 + e = ffaff4d6 -> 11111111101011111111010011010110
h5 = h5 + f = 14aa2e11 -> 00010100101010100010111000010001
h6 = h6 + g = 573232ef -> 01010111001100110011001011101111
h7 = h7 + h = 345501ad -> 00110100010101000000000110101101
```

Этап 8. Узнаем конечный результат

Объединяем значения переменных h0-h7:

```
result = h0 strplus h1 strplus h2 strplus h3 strplus h4 strpplus h5 strplus
   h6 strplus h7 =
f0a4b15f3ba0d2f80c9cfcb6985564e3ffaff4d614aa2e11573232ef345501ad
```

Таким образом мы получили хеш строки «miem cs hse»:

```
f0a4b15f3ba0d2f80c9cfcb6985564e3ffaff4d614aa2e11573232ef345501ad
```

2.7 Строение сети блокчейн

Блокчейн можно рассматривать как цифровую цепочку, состоящую из взаимосвязанных блоков, содержащих зашифрованные записи транзакций. Для проверки и подтверждения подлинности каждого блока используется сложная система криптографических механизмов. Эта распределенная сеть гарантирует коллективное одобрение и согласованность данных, обеспечивая безопасность и прозрачность транзакций.

Каждый блок цепи содержит некоторую информацию, такую как: NONCE, хеш предыдущего блока, хеш текущего блока и некоторый список транзакций.

Хеш блока строится на основе всей информации, которую он содержит.

NONCE (number that can be only used once) – число, которое может быть использовано один раз. Это единственный изменяемый элемент блока, один из инструментов протокола.

Процесс создания и записи блоков описывается алгоритмами консенсуса. Рассмотрим их подробнее.

2.8 Правила сети: Протоколы, Алгоритмы консенсуса

Протокол – это свод правил и действий, по которым будут осуществляться записи в реестр в определенной последовательности. Он обеспечивает безопасность транзакций в сети блокчейн.

Алгоритм консенсуса – механизм, который проверяет соблюдение протокола.

Протокол в себя включает следующие логики:

- Узлы могут появляться и исчезать.
- Узлы могут работать не корректно и непредсказуемо, если их не больше половины всех узлов.

2.8.1 Proof-of-Work — Доказательство работы

PoW (Proof-of-Work — Доказательство работы) – алгоритм консенсуса биткоина, подразумевает, что для участия в добавлении блока транзакций в цепочку членам сети нужно решить трудную и ресурсозатратную математическую задачу по нахождению хеша и публично доказать проделанную работу, чтобы избежать обмана одноранговой системы.

Этим занимаются ноды (майнеры): они ищут такой NONCE для определенного блока, чтобы хеш это блока удовлетворял особому правилу. Например, сеть BitCoin требует, чтобы хеш каждого блока начинался с определенного числа нулей.

Недостатки:

- Избыточность — блокчейну характерна избыточность при проведении вычислений (Узлы как правило, воспроизводят одни и те же вычисления многократно)
- Высокое потребление энергии (это относится к PoW , новые протоколы стремятся избавиться от этого недостатка)

2.8.2 Proof-of-Stake — Доказательство доли

PoS (Proof-of-Stake — Доказательство доли) – покрывает недостатки PoW. В данном механизме вычислительная работа заменяется на блокировку криптовалютных активов у пользователя(стейкинг).

Участники сети оставляют в залог некоторое количество монет за возможность проверки блоков и их добавления в сеть. Они называются валидаторами. Между валидаторами случайным образом выбирается один, который проверит правильность транзакций в блоке и добавит его в цепь. Он также получает награду за проделанную работу, однако если валидатор предлагает добавить невалидный блок, он теряет часть своих активов, оставленных в залог, в виде штрафа.

3 Смарт-контракты

3.1 Разница между сетями Bitcoin и Ethereum

Со стороны уровня сложности логики, которая может быть реализована в сети, Bitcoin является цифровым золотом. Так как BitcoinScript – язык, работающий внутри протоколов Bitcoin, плоский язык без ветвлений и циклов. Он завершается с успехом или ошибкой и может быть использован только для переводов монеток.

Однако, в 2014 году Виталий Бутерин опубликовал White Paper эфириума, в которой предложил концепцию блокчейн протокола, который может выполнять произвольный код. Это был полноценный язык программирования под названием Solidity.

Смарт контракт гарантирует, что программа будет выполнена с одним результатам на всех нодах сети. Инновация – возможность описания произвольной изнес логики, начиная от криптовалют и заканчивая почти всем чем угодно.

Смарт-контракт — цифровая структура, представляющая собой совокупность соглашений и протоколов, управляющих выполнением обязательств между сторонами.

Концепция самоисполняющегося цифрового контракта была впервые предложена Ником Сабо. Он дал хороший пример смарт-контракта в виде взаимодействия с торговым автоматом. Человек производит оплату, и автомат выдаёт нужный товар. Эта сделка предполагает приобретение товара за фиксированную стоимость. Уникальной особенностью такого взаимодействия является автоматизированное выполнение условий сделки торговым автоматом: при совершении оплаты механизм немедленно выдает товар.

На смарт-контракте можно реализовать логику цепочки поставок или автоматической выдачи зарплаты подчиненным.

Ethereum подразумевает использование более сложных алгоритмов, но такие смарт-контракты намного затратно проверить на наличие уязвимостей. Это требует высокой квалификации программиста, реализующего всю бизнес-логику.

Блокчейн Ethereum работает со смарт-контрактами на языке программирования Solidity. В следующем разделе рассмотрим как реализовать свой смарт-контракт.

3.2 Написание смарт-контракта на Solidity

Напишем смарт-контракт для решения следующей бизнес-логики: предположим, что у нас есть магазин по продаже автозапчастей; хозяин магазина принимает платежи. После развертывания смарт-контракта в блокчейн сети владелец принимает платежи с покупателей, они складываются на смарт-контракте, после чего продавец может перевести все деньги на свой счет.

Развернем смарт-контракт в тестовой сети. Для его написания будем использовать среду разработки Remix.

Все смарт-контракты начинаются с обозначения лицензии и версии языка Solidity. В нашем случае лицензия будет с открытым кодом.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;
```

Используя ключевое слово contract, создаем в контракте публичную переменную owner - адрес владельца. Также создаем словарь, в котором будут храниться адреса, с которых приходят платежи и соответсвующие значения количества монет.

```
contract Wallet {
    address public owner;
    mapping (address => uint) public payments;
}
```

Далее создаем конструктор, который будет присваивать контракту владельца.

```
    constructor() {
        owner = msg.sender;
    }
```

После создаем публичную функцию payForItem, которая сможет работать с деньгами. Для этого сипользуем ключевое слово payable. Вызывая эту функцию, клиент будет переводить монеты на наш смарт-котракт. Функция будет заполнять словарь payments.

```
function payForItem() public payable {
    payments[msg.sender] = msg.value;
}
```

withdrawAll()- функция которая будет переводить деньги с контракта на кошелек владельца. тип данных address - payable(owner); присваиваем значение адреса конкретного контракта; перевод денег с контракта на кошелек владельца.

```
function withdrawAll() public {
    address payable _to = payable(owner);
    address _thisContract = address(this);
    _to.transfer(_thisContract.balance);
}
```

Итоговый код смарт-котракта:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;

contract Wallet {
    address public owner;
    mapping (address => uint) public payments;

    constructor() {
        owner = msg.sender;
    }

    function payForItem() public payable {
        payments[msg.sender] = msg.value;
    }

    function withdrawAll() public {
        address payable _to = payable(owner);
        address _thisContract = address(this);
        _to.transfer(_thisContract.balance);
    }
}
```

Скомпилируем и протестируем смарт-котнтракт: выберем кошелек и развернем контракт в тестовой блокчейн сети Ethereum.

Рис. 3.1: Сообщение, что компиляция прошла успешно

Функции смарт-контракта:

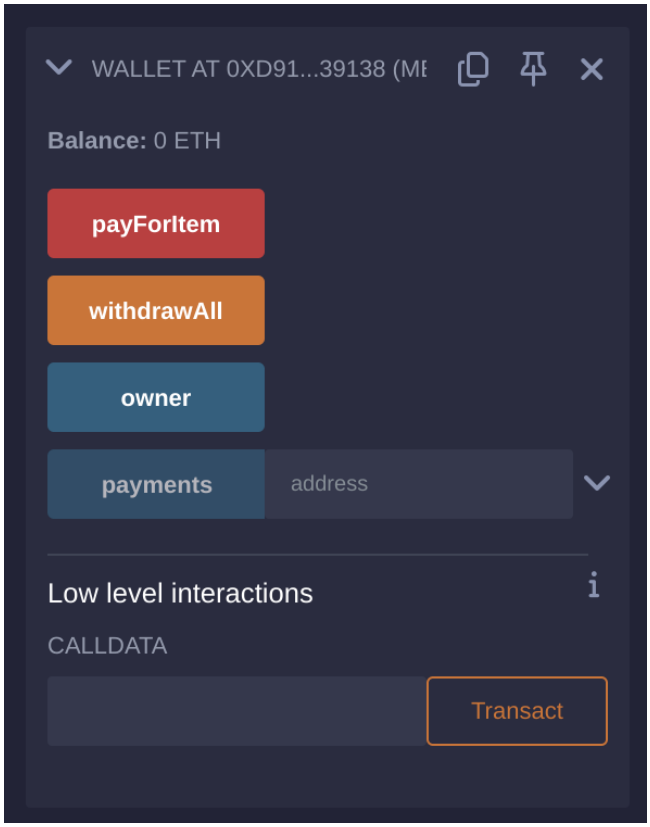

Рис. 3.2: Интерфейс функций контракта

Выбираем аккаунт. В графе VALUE ставим значение 1 Ether и вызываем функцию payForItem.

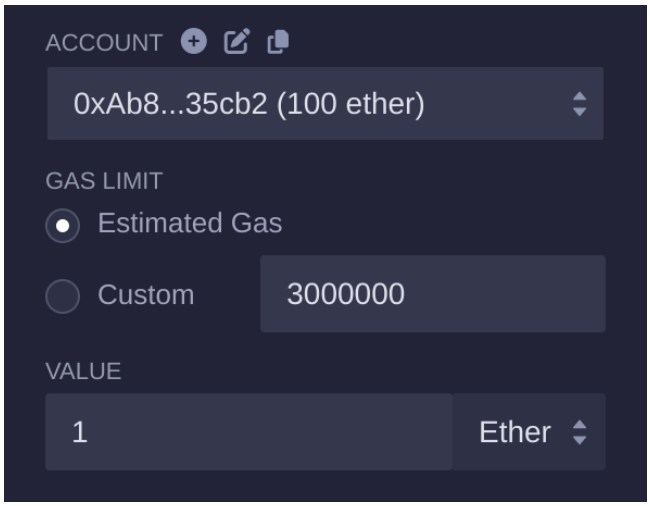

Рис. 3.3: Выбор аккаунта и значения монет

Видим, что с кошелька списался 1 эфир и небольшая комиссия:

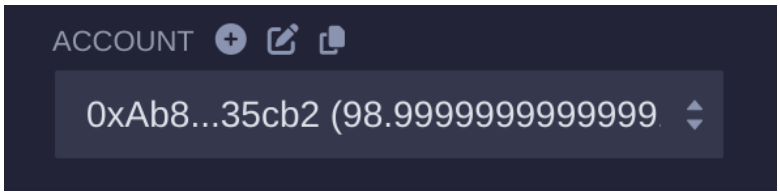

Рис. 3.4: Счет аккаунта клиента

Сообщение, что транзакция прошла успешно:

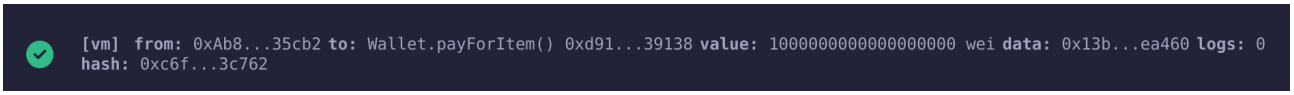

Рис. 3.5: Успешный перевод монет

Теперь вызовем функцию перевода монет на счет владельца withdrawAll.

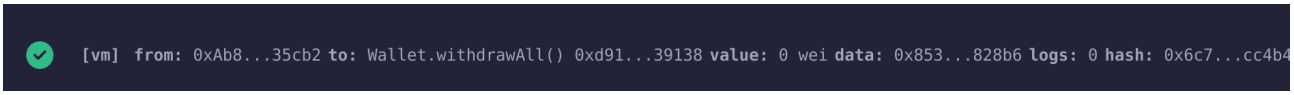

Рис. 3.6: Сообщение, что монеты успешно переведены на счет владельца

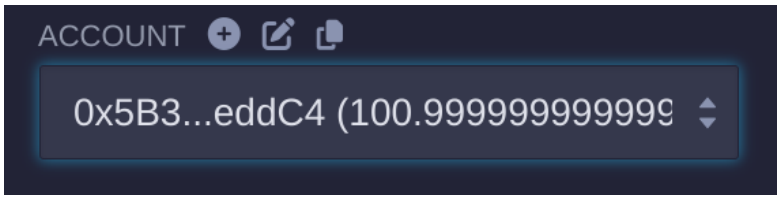

Рис. 3.7: Счет аккаунта владельца

Видим, что транзакция прошла успешно и счет аккаунта владельца увеличился на одну единицу эфира.

Итак, был написан смарт-контракт на языке программирования Solidity и успешно протестирован в тестовой сети Ethereum. В нем была реализована бизнес-логика кошелька магазина, упрощающая оплату товара.

4 Проблемы безопасности

Технология блокчейн имеет несколько проблем безопасности. Рассмотрим их подробно.

Первой проблемой является **Атака 51%**. В случае если количество недобросовестных участников сети превышает половину всех нод, им удается продвинуть свою ложную цепь блоков. Это приводит к уничтожению сети, так как атакающие вносят невалидные транзакции и обнуляют токены, получая выгоду. Таким образом были атакованы блокчейны Ethereum Krypton и Shift в 2016 году.

DDoS-атака является следующей проблемой безопасности. Злоумышленники могут положить ноду большим количеством запросов на добавление их записи в блок. Однако, благодаря распределенности сети и ограниченности размера блока это не приведет к некорректной работе сети.

Главной проблемой безопасности является **появление квантового компьютера**. Асимметричные криптографические шифры, такие как RSA, основаны на предположении о вычислительной сложности факторизации больших чисел. Этот принцип лежит в основе криптографии блокчейна. Однако появление квантовых компьютеров ставит под угрозу эту систему: алгоритмы, предназначенные для квантовых машин, позволяют факторизовать числа за полиномиальное время и взломать сеть.

5 Заключение

- Предоставлен всесторонний обзор блокчейна, включая сравнительный анализ децентрализованных одноранговых сетей и архитектур клиент-сервер, а также различных протоколов достижения консенсуса.
- Определены основные проблемы безопасности технологии блокчейн.
- Написан смарт-контракт для сети блокчейн на языке программирования Solidity.

Применение технологии распределенных реестров (блокчейна) привлекает внимание правительств и исследователей по всему миру, что делает ее предметом активных обсуждений в научных кругах.

6 Список литературы

1) Ethereum Whitepaper [Электронный реурс]. URL: https://ethereum.org/en/whitepaper/ (дата обращения: 29.05.2024)

2) Binance academy Блокчейн [Электронный реурс]. URL: https://academy.binance.com/ru/articles/what-is-blockchain-and-how-does-it-work (дата обращения: 29.05.2024)

3) Bitcoin Whitepaper [Электронный реурс]. URL: https://www.bitcoin.com/satoshi-archive/whitepaper/ (дата обращения: 29.05.2024)

4) Deloitte Global Blockchain Survey 2018 [Электронный ресурс]. URL: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/financial-services/us-fsi-2018-global-blockchain-survey-report.pdf (дата обращения: 29.05.2024)

5) Binance academy Что такое атака 51% [Электронный реурс]. URL: https://academy.binance.com/ru/articles/what-is-a-51-percent-attack (дата обращения: 29.05.2024)

6) Binance academy Sybil атаки [Электронный ресурс]. URL: https://academy.binance.com/ru/articles/sybil-attacks-explained (дата обращения: 29.05.2024)

7) Fernandez-Carames, T. M. Towards PostQuantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks / T. M. Fernandez-Carames, P. Fraga-Lamas // IEEE Access. – 2020. – Vol. 8. – P. 21091-21116. – DOI 10.1109/ACCESS.2020.2968985

8) DEVELOPING INSTRUMENT FOR INVESTIGATION OF BLOCKCHAIN TECHNOLOGY Kushnir D., Kovtsur M., Muthanna A., Kistruga A., Akilov M., Batalov A. Studies in Computational Intelligence. 2022. Т. 1030. С. 123-141.

9) Дрешер, Д. Основы блокчейна / Д. Дрешер. - М.: ДМК Пресс, 2018. - 125 с.