УДК 004.8

А.А. Брилев, Е.Ю. Островская

МЕТОДЫ ВАЛИДАЦИИ ТРАНЗАКЦИЙ ПРИ ПОСТРОЕНИИ БЛОКЧЕЙН

Аннотация. Описаны основные черты технологии блокчейн, принципы ее программной разработки. Проанализированы некоторые аспекты программного создания блокчейна, рассмотрены возможные способы реализации проверки транзакций, описан метод, примененный в биткойне.

Ключевые слова: Блокчейн, биткойн, криптовалюта, транзакция, децентрализованные программы, софт, программирование, код, хэш, C#, разработка, технология, сеть, майнинг, валидация, unspent input, схема, explained, bitcoin, transaction, wallet.

Слово блокчейн в последние месяцы вызвало ажиотаж среди программистов, инвесторов, а затем и среди широкой публики, следящей за новостями в мире электронных новаций. И хотя львиная доля этого интереса спровоцирована спекулятивным ростом цены биткойна, существует консенсус как среди финансистов так и среди экспертов-программистов и инженеров, что блокчейн является прорывной технологией, которая получит свое применение в многих сферах жизни общества и человека.

Суть технологии блокчейн. В общем виде, блокчейн – структура данных - список или связанный список блоков (List или Linked List в терминологии языка C#), построенная с соблюдением ряда правил. Блок блокчейна – это объект, который хранит данные, а также ряд других параметров. Данные, которые хранятся в блоке - называют транзакциями.

В наиболее общем виде блок включает в себя транзакции (одну, две, множество, и по разному реализуется для разных задач), отметку времени создания (timestamp), и предыдущий блок. Очевидно, что при попытке реализации такой концепции, каждый следующий блок будет в двое больше предыдущего и по мере роста блокчейна он очень быстро выйдет за разумные пределы, которые позволяют обмен

информацией в сети. Поэтому, одной из характерных составляющих технологии является хэширование информации.

Любые данные (числа, объекты, символы, строки) могут быть приведены к единому формату, как правило в программировании это приведение к строке. Любая строка произвольной длины может быть преобразована в набор символов фиксированной длины (битовую строку) алгоритмами хэширования. Если к входной строке, независимо от ее длины, будет применен один и тот же алгоритм хэширования, то результатом функции будет выходная строка всегда одинаковой длины. Но, изменение хотя бы одного символа входной строки, будет иметь результатом иную выходную строку. Благодаря этому, в следующий блок можно включать не весь предыдущий блок целиком, а лишь его хэш. Попытка изменения данных предыдущего блока, приводит к изменению хэша, и, таким образом, вынуждает изменять данные всех последующих блоков. На рисунке 1 представлена структура блока и построение блокчейна.

Рисунок 1 - Структура блока и построение блокчейна

Программный код приведенный ниже описывает метод создания блока:

```
public Block(int i, Transaction transactionData, string prev = "")
    {
      index = i;
      Data = transactionData;
      timestamp = DateTime.Now;
      PreviousHash = prev;
      Hash = CalculateHash();
    }
```

где ключевой метод – CalculateHash() – может быть реализован следующим образом:

```
public string CalculateHash()
    {
      StringBuilder sb = new StringBuilder();
      string toEncrypt = index.ToString() + PreviousHash + timestamp.ToString() +
Data.ToString() + nonce;
      using (SHA256 hash = SHA256Managed.Create())
```

 ISSN 1562-9945

```
        {
            Encoding enc = Encoding.UTF8;
            Byte[] result = hash.ComputeHash(enc.GetBytes(toEncrypt));
            foreach(Byte b in result)
            {
                sb.Append(b.ToString("x2"));
            }
        }
        return sb.ToString();
    }
```

Вторым ключевым принципом, на котором строятся большинство проектов по технологии блокчейн, является децентрализация хранения данных. Копии данных хранятся на большом количестве узлов. Изменение данных на одном из них не приводят к изменению данных на остальных. Как только будет найдено несоответствие, узел, данные которого были изменены, восстановит исходные данные путем синхронизации с остальными узлами. Вероятность взлома и одновременного и согласованного искажения данных на множестве (всех) узлах, значительно менее вероятна, и тем меньше, чем больше сеть участников.

Принято выделять три основных типа участников: узлы, майнеры, пользователи.

Узлы хранят данные и постоянно транслируют их по сети. Пользователи – это участники, которые создают транзакции. Майнеры – создают блоки. За этим типом закрепилось такое название прежде всего из-за того, что блокчейн стал уже синонимом криптовалют, а в большинстве криптовалют как раз реализован принцип proof-of-work, задачей которого является недопущение неконтролируемой эмиссии блоков. Для создания блока необходимо произвести вычислительные операции высокой сложности, требующие изрядных ресурсов. В общем случае, майнеры – не более чем узлы, которые обслуживают блокчейн.

Пользователи делают транзакции (необязательно денежные) - это могут быть договора, голосование, просто создание каких-то файлов, и многое другое. Узлы получают транзакции и транслируют их. Майнуры берут транзакции и создают блоки, в которые включены эти транзакции, и, передают блоки ближайшим узлам. Узел, получив новый блок, проверяет его корректность, присоединяет к блокчейну и транслирует обновленный блокчейн соседним узлам. На рисунке 2 представлена общая схема взаимодействия участников сети.

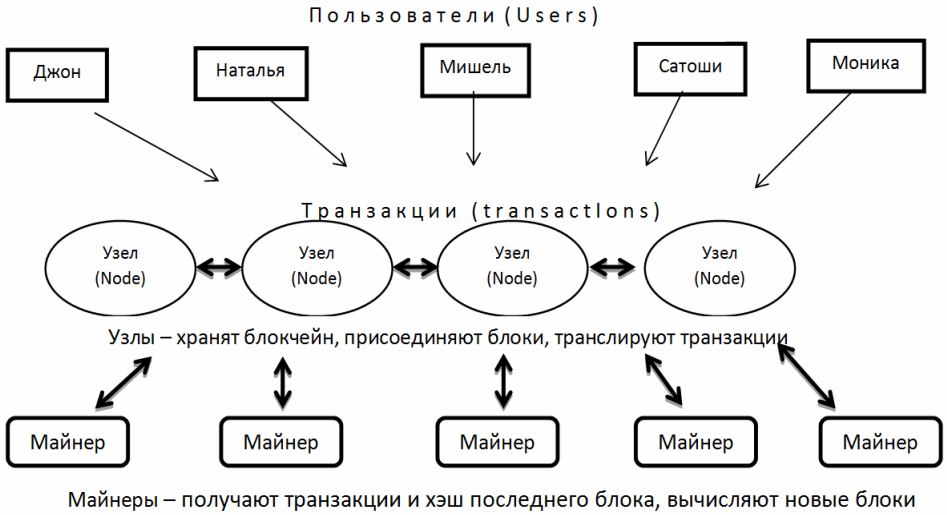

Рисунок 2 - Общая схема взаимодействия участников сети

Одной из первых проблем, которую требуется решить разработчику блокчейн проекта, является стратегия и алгоритм валидации - проверки корректности - транзакций.

Транзакция в упрощенном виде имеет пять полей (свойств): адрес отправителя, адрес получателя, сумму и отметку времени. Для удобства валидации, может быть также введено поле TransactionHash, которое будет содержать хэш предыдущих полей.

```
public class Transaction
{
    public string Sender { get; set; }
    public string Recipient { get; set; }
    public decimal Amount { get; set; }
    public DateTime Timestamp { get; set; }
    public string TransacationHash { get; set; }
}
```

Валидация транзакции, должная включать проверку трех основных параметров: существование получателя, достаточность средств у отправителя и отсутствие данной транзакции среди уже существующих.

Для организации проверки транзакций, требуется решить где и как будут храниться данные пользователей (счета пользователей, которые также обычно представляют собой битовую строку фиксированной длины), и, данные совершенных транзакций. В процессе разработки мною были апробированы следующие варианты.

1. Часть узлов выполняют функцию «бухгалтера»: содержат список зарегистрированных счетов и, отдельно, список всех совер-

 ISSN 1562-9945

шенных и подтвержденных транзакций. Эти узлы также постоянно синхронизируются между собой.

2. Зарегистрированные счета включаются в блокчейн. Каждый блок несет в себе в части данных адреса счетов.

Преимущества первого способа состоят в том, что сам блокчейн разгружается, каждый блок будет меньше по размеру, и, собственно проверка корректности транзакции будет проще, программный код – короче. Недостатки – в том, что появляются дополнительные объекты, которые нужно синхронизировать, продумывать устойчивость от атак, потенциально уменьшается степень децентрализации.

Второй способ более изысканный, однако, потребует более сложных методов проверки, и, приведет к некоторому увеличению размера блока.

Итак, корректность проверки транзакции начинается с проверки существования получателя. Если данные зарегистрированных счетов хранятся в отдельном списке, то этот список просто перебирается до тех пор, пока не будет найден соответствующий счет. Если данные хранятся только в блокчейне, то узел перебирает весь блокчейн пока не найдет совпадение.

```
bool IsValidAddresses(Transaction someTransaction, List<Account> ListOfAllAccounts)
{
        if (someTransaction.Sender == someTransaction.Recipient) return false;
        bool isValidSender = false;
        bool isValidRecipient = false;
        foreach(Account account in ListOfAllAccounts)
        {
           if (someTransaction.Sender == account.Address) isValidSender = true;
           if (someTransaction.Recipient == account.Address) isValidRecipient = true;
        }
        return (isValidRecipient && isValidSender);
}
```

Интереснее реализуется задача проверки достаточности средств. Когда вы пользуетесь традиционными банковскими услугами, вы всегда видите остаток на счету. Соответственно, для реализации мошеннической транзакции (отправки средств, превышающих остаток), достаточно иметь возможность изменить одну цифру. Такая угроза противоречит всей сущности блокчейна. Для более надежного отслеживания остатка средств можно получить суммы всех входящих и всех исходящих транзакций, получателем и отправителем, соответственно, каждой будет данный пользователь. Положительная разница между первой, второй суммами и суммой совершаемой сейчас тран-

закции и будет критерием корректности транзакции. Такой способ потребует перебора всего массива прошлых транзакций, будь то в блоках, либо в отдельно хранящемся списке транзакций.

```
public bool IsValidTransaction(Transaction someTransaction, Hashtable transactions, List<Account> ListOfAllAccounts)
{
    if (!IsValidAddresses(someTransaction, ListOfAllAccounts)) return false;
    decimal amountRecieved = 0;
    foreach(Transaction instance in TransactionsAsRecipent(someTransaction, transactions))
    {
      amountRecieved += instance.Amount;
    }
    decimal amountSent = 0;
    foreach (Transaction instance in TransactionsAsSender(someTransaction, transactions))
    {
      amountSent += instance.Amount;
    }
    return ((amountRecieved - amountSent - someTransaction.Amount) >= 0);
}
```

Интересный и изысканный метод проверки доступных средств реализован в коде биткойна. Каждая транзакция имеет две составляющие: input и output. Output хранит данные, касающиеся отправки средств, то есть адреса получателей и суммы. Input ссылается на одну или более транзакций, в которой данный пользователь был получателем средств. Сумма средств в output должна быть равна сумме средств в input. Если output меньше, чем input, то разница отправляется пользователем самому себе.

Любая транзакция, в которой пользователь был получателем средств, становится так называемым unspent input (непотраченный приход) для данного получателя. Ссылки на все unspent inputs хранит его электронный кашелек (wallet). Когда пользователь формирует отправку средств, исходя из суммы, которую нужно отправить, подбирается ближайший по сумме (но превышающий ее) unspent input. Если единичного input не хватает (не существует unspent input с суммой, достаточной для формирования перевода), то подбираются несколько unspent inputs. Все они будут указаны в input формируемой транзакции. В output будет информация о том, кому переводятся средства и сумма. После совершения такой транзакции, все unspent inputs, которые в нее были включены, станут потраченным (spent inputs) и более не смогут быть включены в input следующей транзакции. Одновременно, во-первых, для получателя

 ISSN 1562-9945

средств появится unspent input в сумме данного перевода, и для отправителя появится unspent input в размере сдачи.

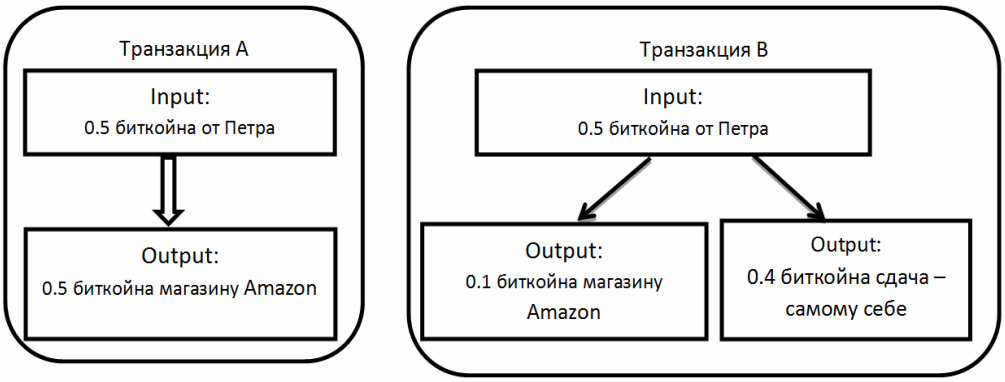

Рисунок 3 - Простейшая транзакция и транзакция со сдачей (change)

Таким образом, реализована, самодостаточная транзакция, транзакция, которая в самой себе хранит информацию о достаточности средств у отправителя. Более того, это позволяет прослеживать историю всей жизни каждой криптомонеты данной платежной системы: когда она появилась, в каких транзакциях участвовала. Этот же алгоритм решает проблему двойного расходования средств.

Однако для такого построения транзакции понадобилось ввести понятие «Wallet» – программа, которая хранит ссылки на все unspent inputs данного пользователя, и, которая позволяет формировать транзакцию при условии подтверждения прав собственности особым криптоключем, известным пользователю. Тем не менее, большинство крупных проектов криптовалют реализуют эту часть проекта именно этим путем. Но эти кошельки на данный момент не в полной мере удобны для пользователя, требовательны к системным ресурсам и нет полной уверенности в степени их безопасности. Поэтому, это перспективная сфера приложения сил нынешнему поколение разработчиков.

ЛИТЕРАТУРА

1. Tapscott Don, Tapscott Alex. Blockchain revolution. New York, 2016
2. Antonopoulos Andreas M. Mastering Bitcoin. Programming the open blockchain. New York, 2017
3. Vigna Paul, Casey Michael J. The Age of Cryptocurrency: How Bitcoin and the Blockchain Are Challenging the Global Economic Order. New York, 2016
4. Wattenhover Roger. The Science of the Blockchain. New York, 2016
5. Pease Marshall, Shostak Robert. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems. 1982
6. Satoshi Nakamoto. Bitcoin:A Peer-to-Peer Electronic Cash System. Bitcoin whitepaper at https://bitcoin.org/bitcoin.pdf
7. Dorier Nicolas. Blockchain Programming in C#. https:// programmingblockchain. gitbooks.io/