Technical Paper

(Информация может быть добавлена)

Децентрализованная финансовая система

CREDITS

Версия 1.5/12.09.2017

Оглавление:

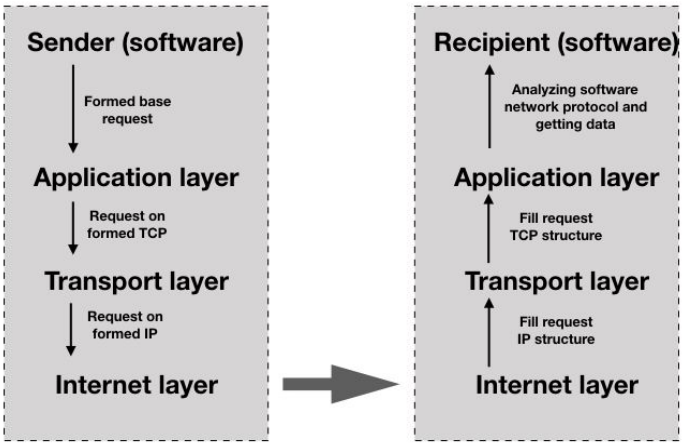
Пиринговая сеть

Пиринговая сеть - одноранговая компьютерная сеть, работающая через интернет и использующая протокол TCP/IP как основной транспортный протокол для передачи данных и команд внутри сети. Стек протоколов TCP/IP включает в себя четыре уровня:

- прикладной уровень (application layer),
- транспортный уровень (transport layer),
- сетевой уровень (межсетевой) (Internet layer),
- канальный уровень (link layer).

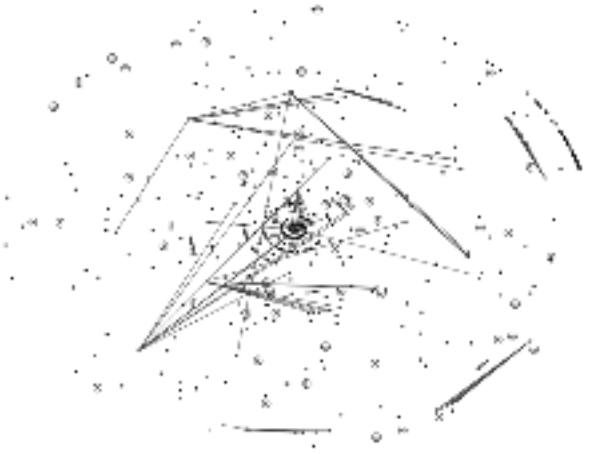
Протоколы этих уровней полностью реализуют функциональные возможности модели OSI. На стеке протоколов TCP/IP построено всё взаимодействие пользователей в IP-сетях. Стек является независимым от физической среды передачи данных, благодаря чему, в частности, обеспечивается полностью прозрачное взаимодействие между проводными и беспроводными сетями.

Для работы пиринговой сети платформы Credits используются:



1. Сетевой уровень. Изначально разработан для передачи данных из одной сети в другую. Пакеты сетевого протокола IP содержат код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный *IP-номер протокола*.
2. Транспортный уровень. Протоколы транспортного уровня (Transport layer) могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных. В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные. TCP (IP идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный поток данных, дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности.
3. Прикладной уровень. На данном уровне происходит обмен данными между программным обеспечением сети.

Для постоянного и стабильного соединения между узлами используется понятие DHT (***d**istributed **h**ash **t**able - распределённая хэш-таблица*). DHT - это класс децентрализованных распределенных

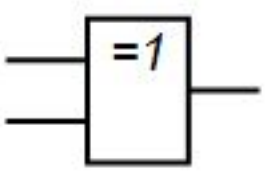


систем поисковой службы, работающей подобно хэш-таблице. Как структура данных, хэш-таблица может представлять собой ассоциативный массив, содержащий пары (ключ-значение). Также, с термином DHT связан ряд принципов и алгоритмов, позволяющих записывать данные, распределяя информацию среди некоторого набора узлов-хранителей, и восстанавливать их, путем распределённого поиска по ключу. Особенностью распределенной таблицы является возможность распределить информацию среди некоторого набора узлов-хранителей таким образом, что каждый участвующий узел смог бы найти значение, ассоциированное с данным ключом. DHT характеризуется следующими свойствами:

- Децентрализация: форма системы коллективных узлов без координации;
- Масштабируемость: система будет одинаково эффективно функционировать при тысячах или миллионах узлов;
- Отказоустойчивость: система будет одинаково надежна (в некотором смысле) с узлами постоянно переключающимися, отключающимися и выдающими ошибки.

Ключевая методика достижения цели заключается в том, что любой узел должен координировать работу только с несколькими узлами в системе — как правило, O(log*n*), где *n* — количество участников (смотри ниже) — так, чтобы только ограниченный объем работы был сделан для каждого изменения количества участников.

Протокол работы пиринговой сети платформы Credits определяет структуру сети, регулирующей связь между узлами, и способ обмена информацией в ней. Узлы сети общаются между собой по протоколу транспортного уровня TCP. Узлы Credits хранят данные посредством распределённых хеш-таблиц (DHT). В итоге над существующей интернет сетью, построенной на основе протокола IP, создается новая виртуальная сеть, в которой каждый узел обозначается специальным номером («Node ID», представленного в виде hash значения). Узел, который хочет присоединиться к сети, обязан пройти «загрузочную» процедуру (bootstrap process). В этот момент узел должен знать адрес другого узла (полученный от пользователя или взятый из списка), который уже входит в виртуальную сеть. Если подключаемый узел еще не входил в эту сеть, то происходит расчет случайного значения ID, которое не принадлежит никакому узлу. ID используется до момента выхода из сети. Алгоритм работы сети базируется на расчете «расстояния» между узлами путем применения операции исключающее ИЛИ к ID этих узлов.



Исключающее ИЛИ (**Сложе́ние по мо́дулю 2**) - булева функция, а также логическая и битовая операция. В случае двух переменных результат выполнения операции является истинным тогда и только тогда, когда один из аргументов является истинным, а второй ложным. Для функции трех и более переменных результат выполнения операции будет истинным только тогда, когда количество аргументов, равных 1, составляющих текущий набор, — нечетное.

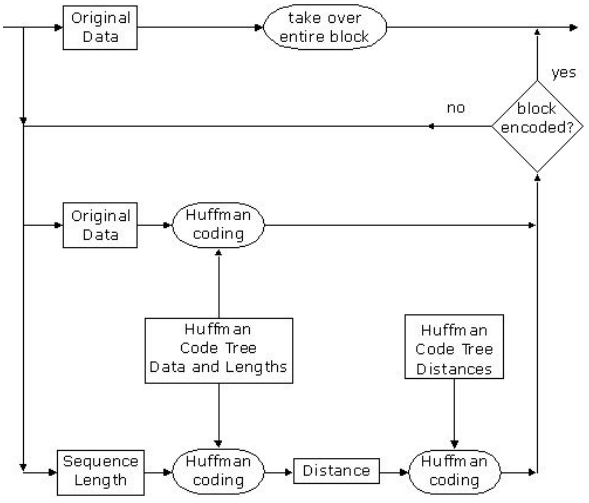
Эта «дистанция» не имеет никакого отношения к географическому положению. К примеру, узлы из Германии и Австралии могут быть «соседними» в виртуальной сети.

Информация в DHT хранится в так называемых «значениях» (values). Каждое «значение» привязано к «ключу» (key).

При поиске соответствующего ключу значения алгоритм исследует сеть в несколько шагов. Каждый шаг приближает к искомому узлу до полного нахождения «значения» либо до отсутствия таких узлов. Количество контактируемых узлов зависит от размера сети логарифмически: при увеличении количества участников (number of participants) вдвое количество запросов увеличится всего на один.

Реестр

Под реестром подразумевается система хранения данных по проведенным транзакциям (действиям) в системе произведенным в системе исчисления (ценности) аккаунта. Реестр представлен в виде словаря Key=Value, где Key - это уникальное значение, присвоенное записи при регистрации в системе; Value - это содержимое транзакции, на основе которого производится действие в системе.
В реестр добавляется пул транзакций, сформированный из одобренных транзакций на этапе голосования узлов. Пул транзакций - это список одобренных транзакций, представленные в формате динамического словаря.
Реестр, локального расположения, хранится на локальном диске в сжатом (архивированном) формате, произведенным по алгоритму сжатия без потерь под названием Deflate. Deflate — это алгоритм сжатия без потерь, использующий комбинацию алгоритмов LZ77 и Хаффмана.



Deflate-поток содержит серии блоков. Перед каждым блоком находится трёхбитовый заголовок:

- Один бит: флаг последнего блока.
  - 1: блок последний.
  - 0: блок не последний.
- Два бита: метод, с помощью которого были закодированы данные.
  - 00: данные не закодированы (в блоке находятся непосредственно выходные данные).
  - 01: данные закодированы по методу статического Хаффмана.
  - 10: данные закодированы по методу динамического Хаффмана.
  - 11: зарезервированное значение (ошибка).

Большая часть блоков кодируется с помощью метода 10 (динамический Хаффман), который предоставляет оптимизированное дерево кодов Хаффмана для каждого нового блока. Инструкции для создания дерева кодов Хаффмана следуют непосредственно за заголовком блока.

Компрессия выполняется в два этапа:

- замена повторяющихся строк указателями (алгоритм LZ77);

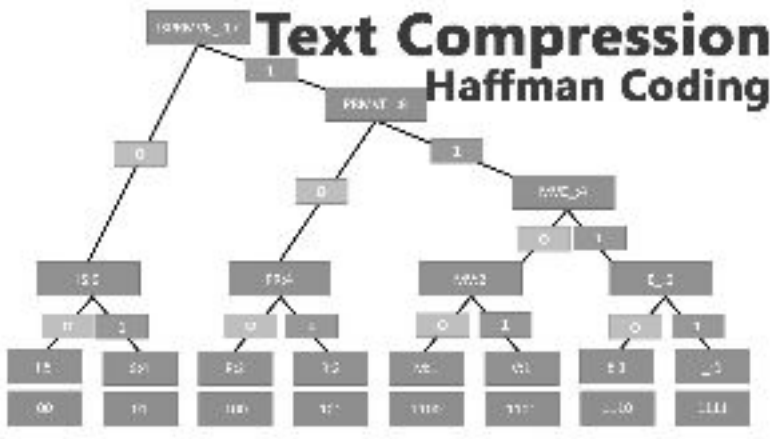
- замена символов новыми символами, основываясь на частоте их использования (алгоритм Хаффмана).

Принцип работы LZ77. Основная идея алгоритма это замена повторного вхождения строки ссылкой на одну из предыдущих позиций вхождения. Для этого используют метод скользящего окна. Скользящее окно можно представить в виде динамической структуры данных, которая организована так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Таким образом, сам процесс сжимающего кодирования согласно LZ77 напоминает написание программы, команды которой позволяют обращаться к элементам скользящего окна, и вместо значений сжимаемой последовательности вставлять ссылки на эти значения в скользящем окне. В стандартном алгоритме LZ77 совпадения строки кодируются парой:

- длина совпадения (match length)
- смещение (offset) или дистанция (distance)

Кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в словаре на значение смещения символов и скопировать значение длины символов, начиная с текущей позиции». Особенность данного алгоритма сжатия заключается в том, что использование кодируемой пары длина-смещение является не только приемлемым, но и эффективным в тех случаях, когда значение длины превышает значение смещения. Пример с командой копирования не совсем очевиден: «Вернуться на 1 символ назад в буфере и скопировать 7 символов, начиная с текущей позиции». Каким образом можно скопировать 7 символов из буфера, когда в настоящий момент в буфере находится только 1 символ? Однако следующая интерпретация кодирующей пары может прояснить ситуацию: каждые 7 последующих символов совпадают (эквивалентны) с 1 символом перед ними. Это означает, что каждый символ можно однозначно определить переместившись назад в буфере, даже если данный символ еще отсутствует в буфере на момент декодирования текущей пары длина-смещение.

Алгоритм Хаффмана — жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Этот метод кодирования состоит из двух основных этапов:



1. Построение оптимального кодового дерева.
2. Построение отображения код-символ на основе построенного дерева.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).[1]

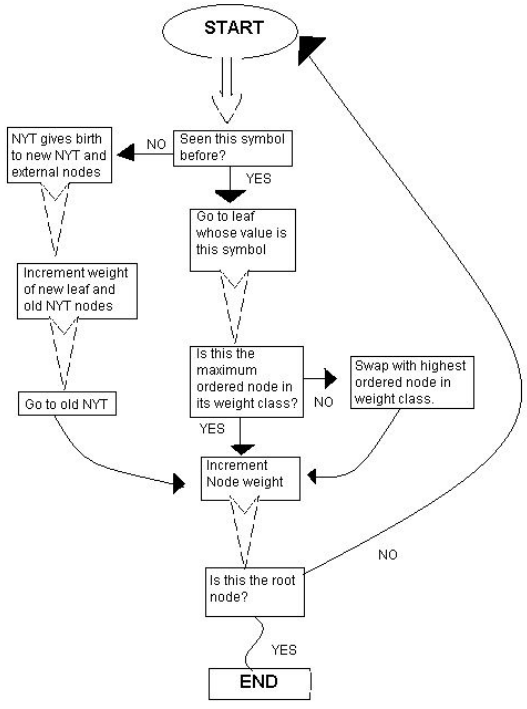
1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Этот процесс можно представить как построение дерева, корень которого — символ с суммой вероятностей объединенных символов, получившийся при объединении символов из последнего шага, его n_0 потомков — символы из предыдущего шага и т. д.

Чтобы определить код для каждого из символов, входящих в сообщение, мы должны пройти путь от листа дерева, соответствующего текущему символу, до его корня, накапливая биты при перемещении по ветвям дерева (первая ветвь в пути соответствует младшему биту). Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.

Поскольку ни один из полученных кодов не является префиксом другого, они могут быть однозначно декодированы при чтении их из потока. Кроме того, наиболее частый символ сообщения закодирован наименьшим количеством бит, а наиболее редкий символ — наибольшим.

Адаптивное сжатие позволяет не передавать модель сообщения вместе с ним самим и ограничиться одним проходом по сообщению как при кодировании, так и при декодировании.



В создании алгоритма адаптивного кодирования Хаффмана наибольшие сложности возникают при разработке процедуры обновления модели очередным символом. Теоретически можно было бы просто вставить внутрь этой процедуры полное построение дерева кодирования Хаффмана, однако, такой алгоритм сжатия имел бы неприемлемо низкое быстродействие, так как построение Н-дерева — это слишком большая работа, и производить её при обработке каждого символа неразумно. К счастью, существует способ модифицировать

уже существующее Н-дерево так, чтобы отобразить обработку нового символа. Наиболее известными алгоритмами перестроения являются алгоритм Фоллера-Галлагера-Кнута (FGK) и алгоритм Виттера.

Все алгоритмы перестроения дерева при считывании очередного символа, включают в себя две операции:

- Первая — увеличение веса узлов дерева. Вначале увеличиваем вес листа, соответствующего считанному символу, на единицу. Затем увеличиваем вес родителя, чтобы привести его в соответствие с новыми значениями веса потомков. Этот процесс продолжается до тех пор, пока мы не доберемся до корня дерева. Среднее число операций увеличения веса равно среднему количеству битов, необходимых для того, чтобы закодировать символ.
- Вторая операция — перестановка узлов дерева — требуется тогда, когда увеличение веса узла приводит к нарушению свойства упорядоченности, то есть тогда, когда увеличенный вес узла стал больше, чем вес следующего по порядку узла. Если и дальше продолжать обрабатывать увеличение веса, двигаясь к корню дерева, то дерево перестанет быть деревом Хаффмана.

Чтобы сохранить упорядоченность дерева кодирования, алгоритм работает следующим образом. Пусть новый увеличенный вес узла равен W+1. Тогда начинаем двигаться по списку в сторону увеличения веса, пока не найдем последний узел с весом W. Переставим текущий и найденный узлы между собой в списке, восстанавливая таким образом порядок в дереве (при этом родители каждого из узлов тоже изменятся). На этом операция перестановки заканчивается.

После перестановки операция увеличения веса узлов продолжается дальше. Следующий узел, вес которого будет увеличен алгоритмом, — это новый родитель узла, увеличение веса которого вызвало перестановку.

Алгоритм обновления дерева Хаффмана должен быть изменен следующим образом: при увеличении веса нужно проверять его на достижение допустимого максимума. Если мы достигли максимума, то необходимо «масштабировать» вес, обычно разделив вес листьев на целое число, например, 2, а потом пересчитав вес всех остальных узлов.

Однако при делении веса пополам возникает проблема, связанная с тем, что после выполнения этой операции дерево может изменить свою форму. Объясняется это тем, что при делении целых чисел отбрасывается дробная часть.

Правильно организованное дерево Хаффмана после масштабирования может иметь форму, значительно отличающуюся от исходной. Это происходит потому, что масштабирование приводит к потере точности статистики. Но со сбором новой статистики последствия этих «ошибок» практически сходят на нет.

Синхронизация данных производится путем обмена пулом транзакций, который рассылается главным узлом, после окончания раунда федеративного голосования по составлению белого списка транзакций.
Формат данных для синхронизации - JSON.
Ниже представлена модель данных, с необходимыми полями, для реализации процесса аналитики и голосования в сети, необходимую для объявления транзакции

```
1. Class CTransaction {
2. HashCode TransactionNumber;
3. CTransactionValue Value;
4. }
5.
6. Class CTransactionValue {
7. String TransactionSender;
8. String TransactionRecipient;
9. Uint TrancationCount;
10. String TransactionCurrence;
11. }
```

Для синхронизации данных используется стандартизированный формат JSON (http://json-rpc.org):

```
{
  {
    «TransactionNumber»:«ctx_KRrYCmZC3Q0pQWjqhCGG66swmTBUK03f»,
    «Value»:{
      «TransactionSender»:«CSxbQDFpHgtbf9SwSdTUatLWDcbiUnTv9P4fFYw29Ab»
      «TransactionRecipient»:«CSxby1nrisp2BldqCT4UluGpfdFnurgCmZB96sMagAF»
      «TrancationCount»:10
      «TransactionCurrence»:«CS»
    }
  },
  {
    «TransactionNumber»:«ctx_KRrYCmZC3Q0pQWjqhCGG66swmTBUK03f»,
    «Value»:{
      «TransactionSender»:«CSxbQDFpHgtbf9SwSdTUatLWDcbiUnTv9P4fFYw29Ab»
      «TransactionRecipient»:«CSxby1nrisp2BldqCT4UluGpfdFnurgCmZB96sMagAF»
      «TrancationCount»:10
      «TransactionCurrence»:«CS»
    }
  }
}
```

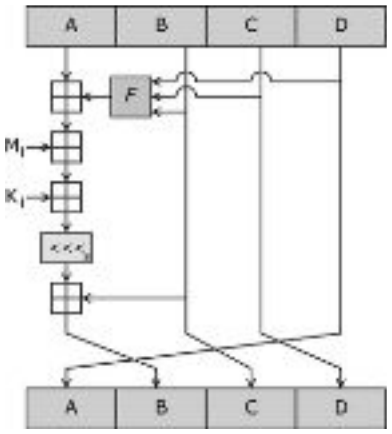
Узлы

Узлом сети называется компьютер, на котором установлены полный клиент сети и хранилище реестра, соединенный с общей системой, принимающий участие в раундах голосования за выбор узла обработки и доверенных узлов, подтверждающий/отклоняющий транзакции и сохраняющий их в реестр. Система подразделяет все компьютеры (узлы) на три категории:

- Главный узел - узел, обеспечивающий анализ и подтверждение транзакций (белого списка), добавление транзакций в реестр

- Доверенный узел - узел, обеспечивающий анализ транзакций и составление первичного белого списка
- обычный узел - узел, участвующий в выборе главного и доверенных узлов.

Каждый узел в системе представлен уникальным кодом являющимся результатом вычисления MD5 hash функции.



MD5 - 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности. На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в *L*. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Ниже приведены 5 шагов алгоритма:

Шаг 1. Выравнивание потока

Сначала дописывают единичный бит в конец потока (байт 80h), затем необходимое число нулевых бит. Входные данные выравниваются так, чтобы их новый размер L′ был сравним с 448 по модулю 512, ($L'=512\times N+448$). Выравнивание происходит, даже если длина уже сравнима с 448.

Шаг 2. Добавление длины сообщения

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит 2^{64}-1, то дописывают только младшие биты (эквивалентно взятию по модулю 2^{64}). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

Шаг 3. Инициализация буфера

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (порядок байтов little-endian, сначала младший байт):

A = 01 23 45 67; // 67452301h
B = 89 AB CD EF; // EFCDAB89h
C = FE DC BA 98; // 98BADCFEh
D = 76 54 32 10. // 10325476h

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

Определим ещё функции и константы, которые нам понадобятся для вычислений.

- Потребуются 4 функции для четырёх раундов. Введём функции от трёх параметров — слов, результатом также будет слово:

1-й раунд: $FunF(X,Y,Z)=(X \wedge Y) \vee (\neg X \wedge Z)$,

2-й раунд: $FunG(X,Y,Z)=(X \wedge Z) \vee (\neg Z \wedge Y)$,

3-й раунд: $FunH(X,Y,Z)=X \oplus Y \oplus Z$,

4-й раунд: $FunI(X,Y,Z)=Y \oplus (\neg Z \vee X)$,

где $\oplus, \wedge, \vee, \neg$ побитовые логические операции XOR, AND, OR и NOT соответственно.

- Определим таблицу констант T[1…64] — 64-элементная таблица данных, построенная следующим образом: $T[n]=int(232 \cdot |sinn|)$.
- Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива *X* из 16 слов по 32 бита. Все раунды однотипны и имеют вид: [abcd k s i], определяемый как a=b+((a+Fun(b,c,d)+X[k]+T[i])≪s), где *k* — номер 32-битного слова из текущего 512-битного блока сообщения, и …≪s — циклический сдвиг влево на *s* бит полученного 32-битного аргумента. Число *s* задается отдельно для каждого раунда.

Шаг 4. Вычисление в цикле

Заносим в блок данных элемент *n* из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

Этап 1

/* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]
[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]
[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

Этап 2

/* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]
[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]
[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]
[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

Этап 3

/* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]
[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]
[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]
[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

Этап 4

/* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]
[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]
[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]
[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

A = AA + A
B = BB + B
C = CC + C
D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива (n + 1) и повторяем цикл.

Шаг 5. Результат вычислений

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и закончив старшим байтом D, то мы получим MD5-хеш. 1, 0, 15, 34, 17, 18…

Ниже представлена таблица основных отличий и функций всех типов узлов

Параметр	Обычный узел	Доверенный узел	Главный узел
Направление обмена	1. Со всеми обычными узлами пиринговой сети 2. Главными узлами	1. С главным узлом текущего раунда для обмена списками 2. Получение обновлений реестра из пиринговой сети	1. С доверенными узлами текущего раунда, 2. Главными узлами для синхронизации списка транзакций с целью

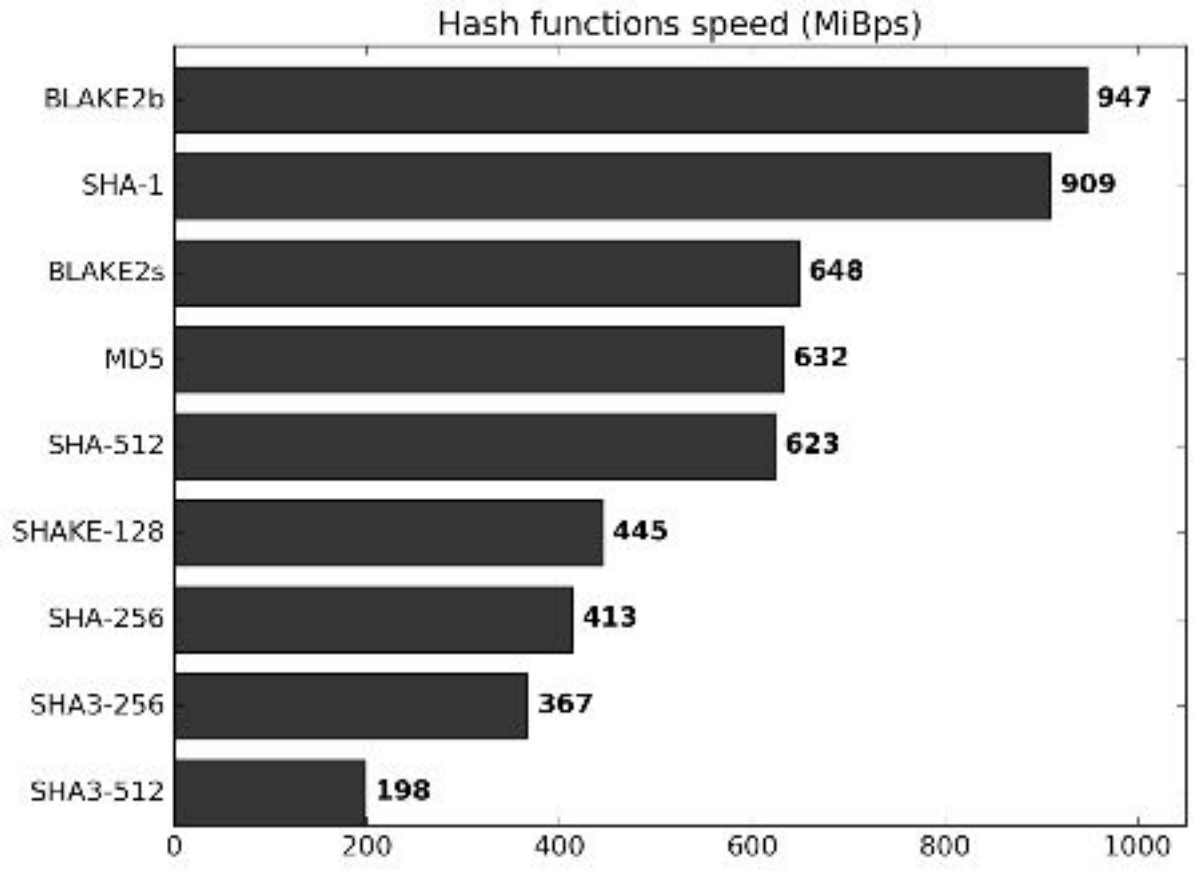
			выявления двойной траты 3. Получение обновлений реестра из пиринговой сети
Данные для обмена	1. Hashcode узла и контрольная сумма реестра для участия в раунде выбора главного и доверенных узлов 2. Hashcode узла для регистрации в сети и получения актуальной версии реестра	1. Hashcode узла и сформированный белый список для отправки на главный узел и составление конечного белого списка 2. Ранжированный список узлов*, допущенных к голосованию за становление доверенным или главным узлом	1. Список транзакций-кандидатов для отправки на доверенные узлы 2. Белый список транзакций для отправки на все узлы сети для добавления в локальные копии реестра вновь сформированного пула транзакций 3. Отфильтрованный список узлов*, допущенных к становлению главным или доверенным узлом для отправки на доверенные узлы данного раунда 4. Сформированный список доверенных узлов* и главного узла на следующий раунд
Основные функции	1. Синхронизация локальной копии реестра 2. Формирование контрольной	1. Синхронизация локальной копии реестра 2. Формирование списка -	1. Синхронизаций локальной копии реестра 2. Формирование списка -

	суммы реестра для участия в выборах	предложения узлов*, имеющих возможность стать главным или доверенным узлом на следующий раунд и отправка на главный узел 3. Формирование списка транзакций путем проверки пула транзакций, отправленного главным узлом, и подтверждение или отклонение транзакций из пула и отправка на главный узел	предложения узлов*, имеющих возможность стать главным или доверенным узлом на следующий раунд и отправка их на доверенные узлы текущего раунда 3. Формирование конечного списка узлов*, содержащий перечень доверенных узлов и главный узел для работы в следующем раунде 4. Формирование списка кандидатов транзакций, для подтверждения / отклонения для записи в реестр т отправка на доверенные узлы текущего раунда 5. Синхронизация с предыдущими транзакциями, обрабатываемыми другими главными узлами для исключения возможности двойной траты 6. обработка сформированн

			ых списков от доверенных узлов и формирование конечного белого списка для добавления в реестр по всей сети 7. Обработка входящих запросов от других главных узлов на проверку транзакций и предоставление информации о других переводах с/на аккаунт

* - под списком узлов понимается список, содержащий hashcode узлов

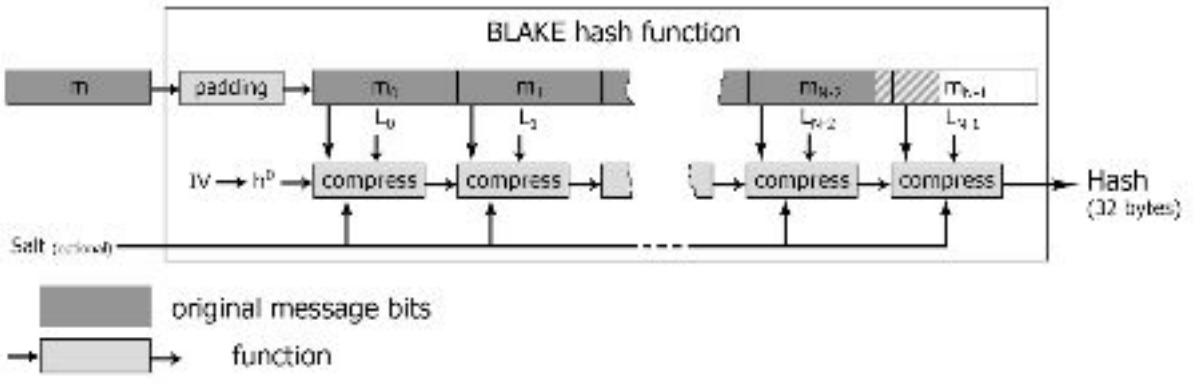
Так как сеть является общественной (Public), то существует вероятность добавления сеть не легализованных узлов (узлов направленных на изменение вероятности принятия решения при решении консенсуса в пользу {конкретное значение}. Для минимизации такой вероятности применяется метод валидации узлов.
Валидация узла - метод (процесс), доказывающий что узел реален и располагает искомыми ресурсами для обеспечения работы сети. Узел подтверждает свою работоспособность путем расчета контрольной суммы хранящегося локально у него реестра и отправка в сеть для последующего участия в голосовании по выбору доверенных и главного узлов в новом руанде по сигналу в сети о начале новых выборов (нового раунда). Поиск контрольной суммы производится по алгоритму blake2s. Сравнение по скорости нахождения результата работы Hash функции с другими алгоритмами представлен ниже.



Хеш-функция BLAKE построена из трёх ранее изученных компонентов:

- режим итерации HAIFA обеспечивает стойкость к атакам второго рода
- внутренняя структура local wide-pipe обеспечивает защиту от коллизий
- алгоритм сжатия для BLAKE является модифицированной версией хорошо параллелизируемого поточного шифра ChaCha, чья безопасность тщательно проанализирована

Процесс хеширования представлен наглядно на блок-схеме:



В BLAKE2 нет добавления констант в раундовой функции. Также изменены константы сдвига, упрощено добавление, добавлен блок параметров, который складывается с инициализирующими векторами. Кроме того, сокращено число раундов с 16 до 12 у функции

BLAKE2b (аналог BLAKE-512) и с 14 до 10 у BLAKE2s (аналог BLAKE-256). В результате число тактов на бит сократилось с 7.49 для BLAKE-256 и 5.64 для BLAKE-512 до 5.34 и 3.32 для Blake2s и Blake2b соответственно.

Ниже представлен результат выполнения Hash расчета для параметров в скобках для данного алгоритма:

BLAKE2s-256("") = 69217A3079908094E11121D042354A7C1F55B6482CA1A51E1B250DFD1ED0EEF9

BLAKE2s-256("The quick brown fox jumps over the lazy dog") = 606BEEEC743CCBEFF6CBCDF5D5302AA855C256C29B88C8ED331EA1A6BF3C8812

BLAKE2s-128("") = 64550D6FFE2C0A01A14ABA1EADE0200C

BLAKE2s-128("The quick brown fox jumps over the lazy dog") = 96FD07258925748A0D2FB1C8A1167A73

Решение консенсуса

Консенсус CREDITS - это метод группового принятия решений, целью которого является приход к выработке окончательных решений, приемлемым для всех узлов сети.

Сравнение консенсусов

Для сравнения различных видов консенсуса определим принципы децентрализованного реестра CREDITS:

● доступность реестра (узлы могут записывать данные в реестр и читать их из него в любой момент времени);

● модифицируемость всеми участвующими узлами в сети;

● согласованность всех узлов системы (все узлы видят абсолютно одинаковую версию реестра, которая обновляется после изменений);● устойчивость к разделению (если один узел становится неработоспособным, это никак не отражается на работе всего реестра).

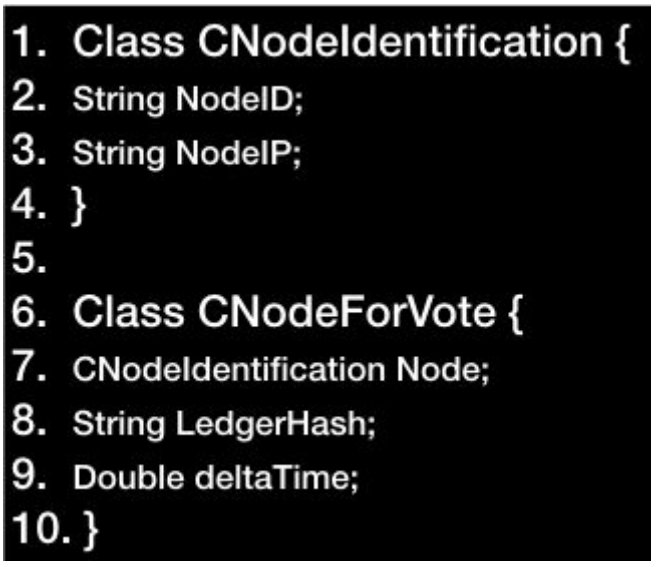
Сравниваемый параметр	Credits specific DPOS	PoW	PoS
Принцип выявления узла, создавшего блок.	Расчет математической функции. Подтверждение хранения последней копии реестра.	Выполнение итеративного расчета математической функции, с изменяющейся сложностью.	Поиск среди участников (конкурирующих узлов) максимального стека.

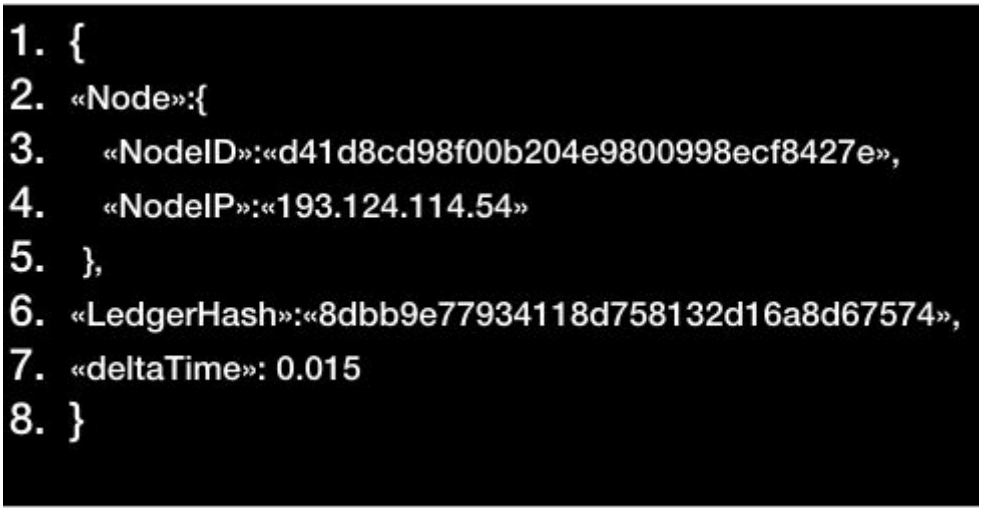
Атака 51%.	Маловероятна, так как необходимо иметь полный реестр на ресурсах и вычислительную мощность для расчета, а выбор доверенных узлов происходит динамически.	Вероятна, но очень дорого обойдется по использованию ресурсов.	Вероятна, но дорого стоит, изза необходимости увеличить свой собственный стек.
Компенсация за проделанную работу на узле для добавления в реестр/блокчейн.	Рассчитывается автоматически, зависит от комиссии за операцию.	Предоставляется фиксировано за майнинг блока.	Предоставляется фиксировано за майнинг блока.

Процесс поиска главного и доверенного узлов

Процесс работы системы принятия решений о добавлении транзакций в общий децентрализованный реестр можно представить в виде итерационного (циклического) подхода. Цикл (раунд) может быть представлен по следующему алгоритму:

1. Расчет контрольной суммы (алгоритм BLAKE2s, описано выше) реестра на узлах, не участвующих в предыдущих раундах (итерациях) и отправка результата в сеть для участия в выборах. Модель данных и формат представлены ниже.





2. формируется первичный список на главном узле последнего раунда и отправляется доверенным для присвоения ранжирования для каждого узла
3. На доверенном узле происходит присвоение каждому узлу числовое значение, полученное на основе выполнения функции по получению случайного значения:

```
boost::random::mt19937 gen;
boost::random::uniform_int_distribution<> dist(0, [count_list_nodes-1]);
int x = dist(gen);
```

4. Обработка и формирование ранжированного списка узлов на основе списков, полученных от доверенных узлов последнего раунда, сортировка списка по уменьшению присвоенного ранга и исключение из списка, тех которые находятся ниже максимально допустимого количества для данного раунда. Максимальное значение допустимого количества возможных узлов в списке рассчитывается согласно математической модели и обоснования, основываясь на сложности сети и количества узлов (Математическая модель будет представлена в следующей редакции).
5. Главный узел пересортирует список в соответствии с увеличением времени формирования контрольной суммы реестра. Результатом будет являться список узлов во главе с узлом с наименьшим временем формирования контрольной суммы, этот узел и будет являться главным узлом в следующем раунде, все остальные узлы являются доверенными.
6. Главный узел текущего раунда отправляет в сеть сформированный список узлов. Главный узел следующего раунда ожидает подключения доверенных узлов, после чего начинается следующая итерация. Если доверенный узел из списка по какой либо причине не подключается - то узел производит попытку подключения самостоятельно, если попытка не удалась - то узел исключается из списка, и отправляется приглашение следующему узлу, находящемуся за чертой.

Анализ и принятие решения по добавлению транзакции в белый список.

Для анализа и принятия решения по добавлению транзакции в белый список мы предлагаем использовать модификацию алгоритма федеративного голосования, называемую византийские генералы (BTF - Byzantine Fault Tolerant).

Образно задача может быть описана следующим образом:

Византия. В ночь перед великим сражением, Византийская армия содержит n легионов. Каждый из них подчиняется своему генералу. У всей византийской армии есть главнокомандующий, руководящий генералами. Империя находится в упадке и среди генералов, включая главнокомандующего, могут быть предатели. В течение всей ночи, каждый из генералов получает от предводителя приказ о действия на утро. Это может быть один из двух вариантов «атаковать» или «отступать». Если все честные генералы атакуют - они одержат победу. Если все отступят - им удастся сохранить армию. Если часть атакуют, а часть отступят - они терпят поражение. Если главнокомандующий предатель, он может дать разным генералам разные приказы, следовательно, его приказы не стоит выполнять беспрекословно. Если же каждый генерал будет действовать независимо от других, результаты битвы также могут быть плачевными. Поэтому генералы нуждаются в обмене информацией друг с другом, чтобы прийти к соглашению.

Определение

Есть n генералов. Связь между ними осуществляется посредством надежной связи (например, телефон). Из n генералов m являются предателями и пытаются воспрепятствовать соглашению между лояльными генералами. Согласие заключается в том, чтобы все лояльные генералы узнали о численности всех лояльных армий и пришли к одинаковым выводам (путь и ложным) относительно состояния предательских армий. (Последнее условие важно, если генералы на основании полученных данных планируют выработать стратегию и необходимо, чтобы все генералы выработали одинаковую стратегию)

В результате:

Каждый лояльный генерал должен получить вектор длины n, где i-й элемент либо обязательно содержит численность i-ой армии (в том случае, если командир лоялен), либо содержит произвольное число в противном случае. Вектора должны быть полностью одинаковыми у всех лояльных генералов.

Алгоритм решения

Шаг 1: Каждый из генералов посылает остальным сообщение, где указывает численность своей армии. Предатели могут указать различные числа в разных сообщениях, а лояльные указывают истинное количество. Генерал g1 указал 1 (тысяча воинов), генерал g2 - 2, генерал g3 соответственно указал трем остальным генералам x, y, z, генерал g4 - 4. Шаг 2: Каждый из генералов вычисляет свой вектор из информации, полученной от остальных генералов. Получается: vect1 (1,2,x,4), vect2 (1,2,y,4), vect3 (1,2,3,4), vect4(1,2,z,4). Шаг 3: Генералы

посылают свои вектора другим. Генерал g3 вновь посылает произвольные значения. Получаются следующие векторы:

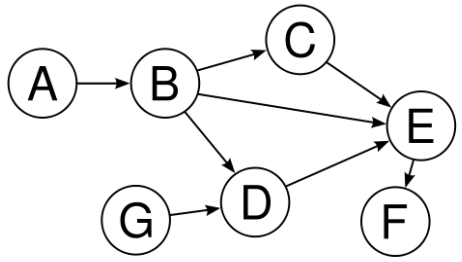
g_1	g_2	g_3	g_4
(1,2,y,4)	(1,2,x,4)	(1,2,x,4)	(1,2,x,4)
(a, b, c, d)	(e, f, g, h)	(1,2,y,4)	(1,2,y,4)
(1,2,z,4)	(1,2,z,4)	(1,2,z,4)	(i, j, k, l)

Шаг 4: Каждый из генералов проверяет каждый элемент в полученных векторах. В том случае, если одно значение совпадает как минимум в двух векторах, оно помещается в результирующий вектор, в противном случае, соответствующий элемент помечается как «неизвестен». В итоге все генералы получат один вектор (1,2, неизвестен, 4). Следовательно, согласие достигнуто. Для n=3 и m=1 согласие достигнуто не будет.

Относительно реально работающей системы алгоритм выглядит так же, только под генералом понимается узел, а сообщения (выстраиваемый вектор) - это пул транзакций, в котором каждой транзакции присваивается значение (1 - подтверждена (Confirm), 0 - отклонена (Fault)).

Поиск участков сделки

Пиринговую сеть Credits можно представить в виде графа, с аккаунтами пользователей в виде вершин и множеством возможных транзакций в виде множества направленных ребер, соединяющие две вершины (аккаунта). Так как все ребра имеют начальную и конечную вершины, значит всегда можно построить ориентированный граф (орграф). Если принять за тождество следующие условия:



- любая транзакция всегда имеет отправителя и получателя
- любая вершина (аккаунт) всегда может быть соединена с другой вершиной (аккаунтом) с помощью направленного ребра(транзакции)
- любая вершина графа (аккаунт) имеет конечное количество направленных ребер (входящих и исходящих транзакций)

тогда мы всегда можем сказать что орграф может содержать искомый маршрут (конечную последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершиной ребром) выполнения необходимых условий транзакции и построить простую цепь (Простая цепь - маршрут в орграфе без повторяющихся вершин). Так как заранее граф не известен, то исходя из теории графов - маршрут придется строить по неизвестному ориентированному графу. Как известно, на классе таких графов длина обхода $\Theta(nm)$, где n – число вершин, а m – число ребер графа. Для любого графа существует обход длиной $O(nm)$ и существуют графы с минимальной длиной обхода $\Omega(nm)$. Обход неизвестного графа означает, что топология графа заранее неизвестна и мы узнаем ее только в процессе движения по графу. В каждой вершине видно, какие ребра из нее исходят, но в какую вершину ведет ребро можно узнать, только пройдя по ней. Это аналогично задаче обхода лабиринта роботом, находящимся внутри него и не имеющем плана лабиринта. Если

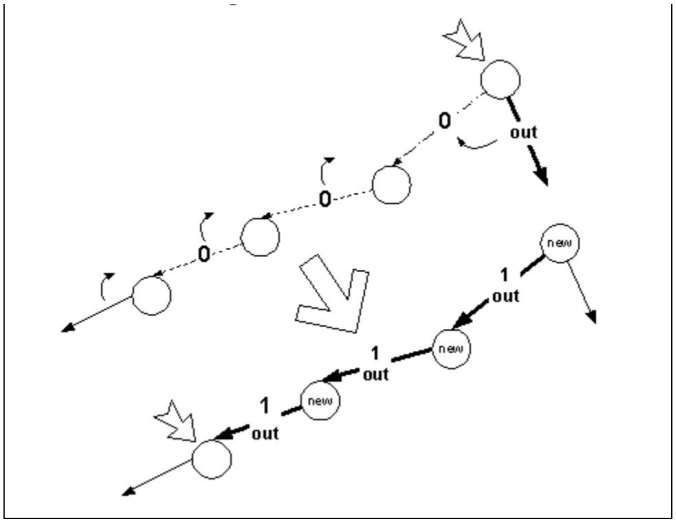
число состояний ограничено, то робот – это конечный автомат. Такой робот является аналогом машины Тьюринга: лента заменена графом, а ее ячейки привязаны к вершинам и ребоам графа. Добавим ограничительное условие - ориентация ребра должна удовлетворять условиям транзакции/контракта, если ориентация не соответствует искомым значениям то переход по ребру не осуществляется.

И так для построения маршрута с простой цепью поставим задачу.

Ориентированный граф, на котором работает робот, можно определить как $G=(V,E,\alpha,\beta,\gamma,\delta,X,\chi)$, где:

- V – множество вершины;
- E – множество ребер (для удобства будем считать, что $E \cap V = \varnothing$);
- $\alpha: E \to V$ – функция, определяющая начальную вершину (начало) ребра;
- $\beta: E \to V$ – функция, определяющая конечную вершину (конец) ребра;
- $\gamma: V \to E$ – функция, определяющая первую дугу в цикле исходящих ребер, с условием:
  - $\forall v \in V \ dout(v)>0 \Rightarrow \alpha(\gamma(v))=v$;
- $\delta: E \to E$ – функция, определяющая следующее ребро в цикле исходящих ребер, с условием:
  - $\forall e \in E \ \exists \ k=0..dout(\alpha(e))-1 \ \delta k(e)=\gamma(\alpha(e))$, где $\delta k=\delta \circ \delta \circ ... \circ \delta$ и знак суперпозиции применяется k-1 раз;
- X – множество символов, которые могут храниться в ячейках вершин и ребер;
- $\chi: V \cup E \to X$ – функция, определяющая символы, хранящиеся в ячейках вершин и ребер.

Граф конечен, если конечны множества V и E. Число вершин конечного графа обозначим $n=|V|$. Два ребра e и e` смежны, если $\beta(e)=\alpha(e`)$. Маршрут или путь (path) – это последовательность смежных ребер. Маршрут проходит через вершину, если она является началом или концом некоторого ребра маршрута. Начало первого ребра маршрута называют началом маршрута, а конец последнего ребра – концом маршрута. Простой путь (simple path) – маршрут, не проходящий через одну вершину более одного раза. Контур (cycle) – маршрут, начало и конце которого совпадают; в простом контуре (simple cycle) начало и конец – единственные совпадающие вершины. Обход – маршрут, содержащий все ребра графа. Граф сильно связен, если любая пара вершин связана некоторым маршрутом.



Робот на графе G определяется как R=(Q,X,T), где:

- Q – множество состояний;
- X – множество входных символов (совпадает с множеством символов в ячейках графа);
- $T \subseteq Q \times X \times X \times Q \times X \times X \times \{i,o\}$ – множество переходов.

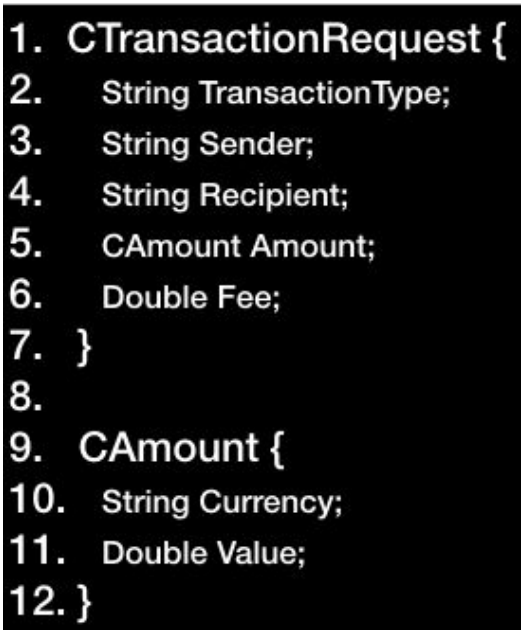
В каждый момент времени робот располагается в текущей вершине $v \in V$ на текущем ребре $e \in E$, исходящей из v, то есть, $\alpha(e)=v$. Робот находится в состоянии $q \in Q$, читает символ вершины $xv=\chi(v)$ и символ ребра $xe=\chi(e)$. Переход $(q,xv,xe,q`,x`v,x`e,i/o) \in T$ означает, что

робот переходит в состояние q`, записывает символы в ячейку вершины $\chi(v)=x`v$ и в ячейку ребра $\chi(e)=x`e$. При внутреннем переходе (i) робот остается в той же вершине v, но переходит на следующее ребро в v-цикле $\delta(e)$. При внешнем переходе (o) робот переходит по ребру e в ее конечную вершину $\beta(v)$ на первое исходящее из нее ребро $\gamma(\beta(v))$. Робот конечен, если множества Q и X конечны. Робот детерминирован, если для каждой тройки $(q,xv,xe)\in Q\times X\times X$ существует не более одного перехода $(q,xv,xe,q`,x`v,x`e,i/o)\in T$. Если для некоторой такой тройки (q,xv,xe) нет ни одного перехода, будем говорить, что робот останавливается в состоянии q в вершине с символом xv на ребре с символом xe. Будем считать, что один символ $\varepsilon\in X$ выделен как начальный символ, который находится во всех ячейках вершин и ребер в начале работы робота. Вершина, с которой робот начинает работать, будем называть начальной и обозначать v0, начальным ребром является первое исходящее ребро $\gamma(v0)$.
Последовательность внешних переходов, которые делает робот R на графе G с начала работы, очевидно, определяет маршрут в G, который мы будем называть пройденным маршрутом. Если робот останавливается, этот маршрут конечен.
Робот обходит граф, если он останавливается на этом графе и пройденный маршрут является обходом. Если каждое ребро, исходящее из вершины v, принадлежит маршруту P, эту вершину будем называть полностью пройденной (в маршруте P). Очевидно, для сильно связных графов пройденный маршрут является обходом тогда и только тогда, когда все его вершины полностью пройдены. Для решения вышеозначенной задачи предлагается использовать алгоритм прохода по не пройденным ребрам: мы находимся в open-вершине v и, если из нее исходит хотя бы одно ребро, то текущее ребро p – первое по v-циклу непройденное ребро. Двигаемся по непройденным ребрам до пройденной вершины или до конечной вершины. Для этого сначала проверяем, не является ли v конечной вершиной. Если вершина v конечная, то возвращаемся из процедуры с ответом “пришли в конечную вершину”. Заметим, что в графе конечная вершина может быть только в том случае, если она единственная вершина графа. В этом случае, очевидно, робот обошел граф. Если вершина v не конечная, из нее исходит непройденное ребро p=(v,w). Помечаем вершину v меткой new, status ребра p меняем с 0 на 1, помечаем его как out-ребро и идем по нему в ее конец w. Если вершина w пройдена, возвращаемся из процедуры с ответом “пройдена хорда”. В противном случае, помечаем ее как root-вершину и повторяем действия для вершины w.

Транзакции

Транзакция - это минимальная единица системы, информирующая платформу о выполнении методов контракта или прямых переводов между аккаунтами без создания смарт контракта, с последующим размещением результата выполнения в пиринговой сети.

Транзакция формируется на клиенте в соответствии с моделью:



```
1.  CTransactionRequest {
2.     String TransactionType;
3.     String Sender;
4.     String Recipient;
5.     CAmount Amount;
6.     Double Fee;
7.   }
8.
9.   CAmount {
10.    String Currency;
11.    Double Value;
12. }
```

В соответствии с вышеуказанной моделью, можно вывести содержимое передаваемых данных:

```
1.   {
2.     «TransactionType»:«PAYMENT»
3.     «Sender»:«CSxbQDFpHgtbf9SwSdTUatLWDcbiUnTv9P4fFYw29Ab»
4.     «Recipient»: «CSxby1nrisp2BIdqCT4UluGpfdFnurgCmZB96sMagAF»
5.     «Amount»: {
6.        «Currency»: «CS»
7.        «Value»: 100
8.     }
9.     «Fee»: 0.01
10.  }
```

После формирования транзакция подписывается PrivateKey (алгоритм ecdsa25519 - описан в разделе безопасности), шифруется по алгоритму гомоморфного шифрования (алгоритм описан в разделе безопасности) и отправляется в сеть для последующей обработки.

После попадания транзакции в сеть формируется ее hash на основе алгоритма Blake2s (описан выше).

Безопасность и возможные угрозы

Для увеличения криптостойкости и надежности системы используется алгоритм гомоморфного шифрования что позволяет добиться повышения скорости анализа и обработки транзакции.

Гомоморфное шифрование — форма шифрования, позволяющая производить определённые математические действия с зашифрованным текстом и получать зашифрованный результат, который соответствует результату операций, выполняемых с открытым текстом. Например, один человек мог бы сложить два зашифрованных числа, а затем другой человек мог бы расшифровать результат, не используя ни одно из них. Гомоморфное шифрование позволило бы объединить в одно целое различные услуги, не предоставляя данные для каждой услуги. Различают частично гомоморфные и полностью гомоморфные криптосистемы. В то время как частично гомоморфная система позволяет производить одновременно только одну из операций — сложение или умножение, полностью гомоморфные криптосистемы поддерживают одновременное выполнение обеих операций, что позволяет гомоморфно вычислять произвольные логические контуры.

Полностью гомоморфное шифрование — шифрование, позволяющее для данного шифротекста $\pi_1,\ldots,\pi_t$ любому (не только держателю ключа) получить шифротекст любой желаемой функции $f(\pi_1,\ldots,\pi_t)$, до тех пор, пока данная функция может быть эффективно вычислена.

Шифротекст, **шифртекст** — результат операции шифрования. Часто также используется вместо термина «криптограмма», хотя последний подчеркивает сам факт передачи сообщения, а не шифрования.

При рассмотрении шифротекста как случайной величины Y=f(X,Z), зависящей от соответствующих случайных величин открытого текста X и ключа шифрования Z, можно определить следующие свойства шифротекста:

- Свойство однозначности шифрования:

$H(Y|XZ)=0$

- Из цепных равенств следует

$H(ZYX)=H(Z)+H(Y|Z)+H(X|YZ)=H(Z)+H(Y|Z)+0$ (из свойства однозначности расшифрования)

$H(ZXY)=H(Z)+H(X|Z)+H(Y|XZ)=H(Z)+H(X)+0$ (из принципа независимости открытого текста от ключа и свойства однозначности шифрования)

тогда

$H(Y|Z)=H(X)$ это равенство используется для вывода формулы расстояния единственности.

- Для абсолютно надежной криптосистемы

$I(Y,X)=0$, то есть $H(Y)=H(Y|X)$

Функция гомоморфного шифрования $E(k,m)$, где m — открытый текст, k — ключ шифрования, гомоморфна относительно операции $*$ *(умножения)* над открытыми текстами, если существует

эффективный алгоритм *M*, который получив на вход любую пару криптограмм вида $E(k,m_1),E(k,m_2)$, выдает криптограмму *C* такую, что при дешифровании с будет получен открытый текст m_1*m_2. Аналогично определяется гомоморфизм относительно операции + *(сложение)*.

В то время, как частично гомоморфные криптосистемы гомоморфны лишь относительно одной операции над открытым текстом (либо сложения или умножения), полностью гомоморфные системы поддерживают гомоморфизм относительно обеих операций (как сложения, так и умножения). То есть для них выполнены следующие условия:

$$\begin{cases} Dec(Enc(m_1) \otimes Enc(m_2)) = m_1 * m_2 \\ Dec(Enc(m_1) \oplus Enc(m_2)) = m_1 + m_2 \end{cases}$$

Более того, гомоморфности по операциям сложения и умножения достаточно, чтобы система была полностью гомоморфной.

Защищенность большинства полностью гомоморфных схем основана на сложности решения проблемы обучения с ошибками. Только у LVT (Криптосистема на NTRU созданная Лопез-Альтом,Тромером и Вайкунтанаханом) схемы защита реализована на варианте вычислительной NTRU задачи. У всех этих систем в отличие от ранних схем более медленное нарастание шума в процессе гомоморфных вычислений. В результате дополнительной оптимизации была получена криптосистема с практически оптимальной асимптотической сложностью: Сложность вычисления *T* операций над зашифрованными данными с параметром защиты *k* имеет сложность вычисления лишь $T \cdot polylog(k)$. Эти оптимизации построены на технике Смарта-Веркаутерена, которая позволяет сжимать набор текстовых переменных в один шифротекст и работать над данными переменными одним потоком. Многие достижения для гомоморфных систем также были использованы в криптосистеме на целых числах.

Схема полностью гомоморфного шифрования может быть рассмотрена на примере вычислений в Z_2.

Шифрование

Процесс шифрования данных можно представить следующим образом:

1. Выбирается произвольное нечетное число $p=2k+1$, являющееся секретным параметром. Пусть $m \in \{0,1\}$.

2. Составляется число $z \in Z_2$ такое, что $z=2r+m$, где *r* — произвольное число. Это значит, что $z=m \bmod 2$.

3. В процессе шифрования всякому m ставится в соответствие число $c=z+pq$, где *q* выбирается произвольно. Таким образом, $c=2r+m+(2k+1)*q$. Легко видеть, что $c \bmod 2=(m+q) \bmod 2$, и, значит, злоумышленник сможет определить только четность выхода шифрования.

Расшифрование

Пусть известны зашифрованное число c и секрет p. Тогда процесс расшифрования данных должен содержать следующие действия:

1.Расшифрование с использованием секретного параметра p: $r=c \bmod p=(z+pq) \bmod p=z \bmod p+(pq) \bmod p$, где $r=c \bmod p$ называется шумом и $z \in (-p/2,p/2)$.

2.Получение исходного зашифрованного бита: $m=r \bmod 2$

Обоснование

Пусть есть два бита $m_1,m_2 \in Z_2$ и им в соответствие поставлена пара чисел $z_1=2r_1+m_1$ и $z_2=2r_2+m_2$. Пусть взят секретный параметр $p=2k+1$ и произведено шифрование данных: $c_1=z_1+pq_1$ и $c_2=z_2+pq_2$.

Вычисляется сумма этих чисел:

$c_1+c_2=z_1+pq_1+z_2+pq_2=z_1+z_2+p(q_1+q_2)=2r_1+m_1+2r_2+m_2+(2k+1)(q_1+q_2)$

Для суммы этих чисел расшифрованным сообщением будет сумма исходных бит m_1,m_2:$((c_1+c_2) \bmod p) \bmod 2=(2(r_1+r_2)+m_1+m_2) \bmod 2=m_1+m_2$.

Но не зная p, расшифровать данные не представляется возможным: $((c_1+c_2) \bmod p) \bmod 2=m_1+m_2+q_1+q_2$.

Аналогично проверяется операция умножения:

$c_1c_2=(z_1+pq_1)(z_2+pq_2)=z_1z_2+p(z_1q_2+z_2q_1)+p_2q_1q_2=(2r_1+m_1)(2r_2+m_2)+(2k+1)((2r_1+m_1)q_2+(2r_2+m_2)q_1)=4r_1r_2+2(r_1m_2+r_2m_1)+m_1m_2+2k(2r_1q_2+m_1q_2+2r_2q_1+m_2q_1)+2r_1q_2+m_1q_2+2r_2q_1+m_2q_1$

К полученным результатам необходимо применить процедуру расшифрования, в результате чего получится следующее:

$((c_1c_2) \bmod p) \bmod 2=(4r_1r_2+2(r_1m_2+r_2m_1)+m_1m_2) \bmod 2=m_1m_2$.

Для получения ключей используются эллиптические кривые построенные на основе ecdsa25519

Эллипти́ческая крива́я над полем K — неособая кубическая кривая на проективной плоскости над K' (алгебраическим замыканием поля K), задаваемая уравнением 3-й степени с коэффициентами из поля K и «точкой на бесконечности». В подходящих аффинных координатах её уравнение приводится к виду:

$y2+a1xy+a3y=x3+a2x2+a4x+a6$.

Концептуальная математическая модель

Если *charK* (характеристика поля K) не равна *2* или *3*, то уравнение с помощью замены координат приводится к канонической форме (форме Вейерштрасса):

$y^2=x^3+ax+b$.

Если *charK=3*, то каноническим видом уравнения является вид:

$y^2=x^3+a_2x^2+a_4x+a_6$.

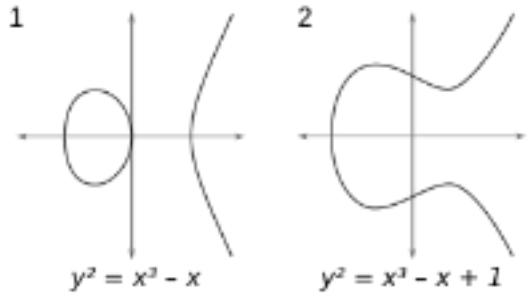
Если *charK=2*, то уравнение приводится к одному из видов:

$y^2+y=x^3+ax+b$ — суперсингулярные кривые

или

$y^2+xy=x^3+ax^2+b$ — несуперсингулярные кривые.

Эллиптические кривые над вещественными числами



Поскольку характеристика поля вещественных чисел — *0*, а не *2* или *3*, то эллиптическая кривая — плоская кривая, определяемая уравнением вида:

$y^2=x^3+ax+b$,

где *a* и *b* — вещественные числа. Этот вид уравнений называется **уравнениями Вейерштрасса**.

Определение эллиптической кривой также требует, чтобы кривая не имела особых точек. Геометрически это значит, что график не должен иметь каспов и самопересечений. Алгебраически, достаточно проверить, что дискриминант

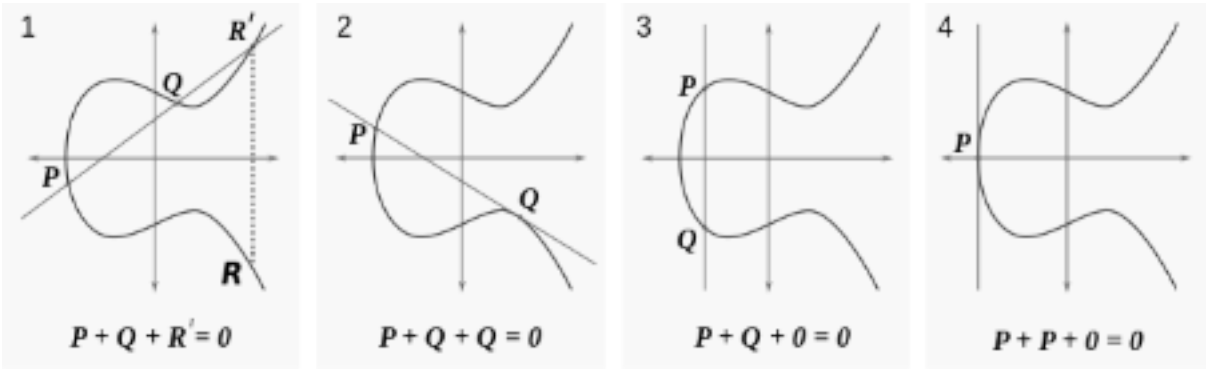
$\Delta=-16(4a^3+27b^2)$

не равен нулю.

Если кривая не имеет особых точек, то ее график имеет две связные компоненты, если дискриминант положителен, и одну — если отрицателен. Например, для графиков выше в первом случае дискриминант равен *64*, а во втором он равен *−368*.

Групповой закон

Добавлением «точки в бесконечности» получается проективный вариант этой кривой. Если *P* и *Q* — две точки на кривой, то возможно единственным образом описать третью точку — точку пересечения данной кривой с прямой, проведенной через *P* и *Q*. Если прямая является касательной к кривой в точке, то такая точка считается дважды. Если прямая параллельна оси ординат, третьей точкой будет точка в бесконечности.



Таким образом, можно ввести групповую операцию *«+»* на кривой со следующими свойствами: точка в бесконечности (обозначаемая символом *O*) является нейтральным элементом группы, и если прямая пересекает данную кривую в точках *P, Q* и *R′*, то *P+Q+R′=O* в группе. Суммой точек *P* и *Q* называется точка *R=P+Q*, которая симметрична точке *R′* относительно оси *Ox*. Можно показать, что относительно введенной таким образом операции лежащие на кривой точки и точка *O* образуют абелеву группу; в частности, свойство ассоциативности операции *«+»* можно доказать, используя теорему о 9 точках на кубической кривой (кубике).

Данная группа может быть описана и алгебраически. Пусть дана кривая $y^2=x^3+ax+b$ над полем *K* (характеристика которого не равна ни *2*, ни *3*), и точки $P=(x_P,y_P)$ и $Q=(x_Q,y_Q)$ на кривой; допустим, что $x_P \neq x_Q$. Пусть $s=(y_P-y_Q)/(x_P-x_Q)$; так как *K* — поле, то *s* строго определено. Тогда мы можем определить $R=P+Q=(x_R,y_R)$ следующим образом:

$x_R=s^2-x_P-x_Q$,

$y_R=-y_P+s(x_P-x_R)$.

Если $x_P=x_Q$, то есть два варианта: если $y_P=-y_Q$, то сумма определена как *0*; значит, обратную точку к любой точке на кривой можно найти, отразив ее относительно оси *Ox*. Если $y_P=y_Q \neq 0$, то $R=P+P=2P=(x_R,y_R)$ определяется так:

$s=(3x^2_P+a)/2y_P$,

$x_R=s^2-2x_P$,

$y_R=-y_P+s(x_P-x_R)$.

Если $y_P=y_Q=0$, то $P+P=O$.

Обратный элемент к точке *P*, обозначаемый $-P$ и такой, что $P+(-P)=0$, в рассмотренной выше группе определяется так:

- Если координата y_P точки $P=(x_P,y_P)$ не равна *0*, то $-P=(x_P,-y_P)$.
- Если $y_P=0$, то $-P=P=(x_P,y_P)$.
- Если $P=O$ — точка на бесконечности, то и $-P=O$.

Точка $Q=nP$, где *n* целое, определяется (при $n>0$) как $Q=P+P\cdots+P$. Если $n<0$, то *Q* есть обратный элемент к $|n|P$. Если $n=0$, то $Q=0 \cdot P=O$. Для примера покажем, как найти точку $Q=4P$: она представляется как $4P=2P+2P$, а точка $2P$ находится по формуле $2P=P+P$.

ecdsa25519 - это криптографическая функция, основанная на эллиптических кривых, разработанная с использованием эллиптической кривой на основе алгоритма Дифи-Хелмана.

Кривая математически обоснована следующей графической функции $y^2 = x^3 + 486662x^2 + x$.

ECDSA25519 сконструирован таким образом, что он избегает многих потенциальных ошибок реализации. По своей конструкции он невосприимчив к тайм-атакам и принимает любую 32-байтную строку в качестве действительного открытого ключа и не требует проверки.

Протокол Ди́ффи-Хе́ллмана на эллиптических кривых — криптографический протокол, позволяющий двум сторонам, имеющим пары открытый/закрытый ключ на эллиптических кривых, получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Этот секретный ключ может быть использован как для шифрования дальнейшего

обмена, так и для формирования нового ключа, который затем может использоваться для последующего обмена информацией с помощью алгоритмов симметричного шифрования. Это вариация протокола Диффи-Хеллмана с использованием эллиптической криптографии.

Описание алгоритма.

Пусть существуют два абонента. Предположим, первый абонент хочет создать общий секретный ключ со вторым, но единственный доступный между ними канал может быть подслушан третьей стороной. Изначально должен быть согласован набор параметров *((p,a,b,G,n,h)* для общего случая и *(m,f(x),a,b,G,n,h)* для поля характеристики 2). Также у каждой стороны должна иметься пара ключей, состоящая из закрытого ключа *d* (случайно выбранное целое число из интервала [*1,n−1*]) и открытого ключа *Q* (где $Q=d \cdot G$ — это результат проделывания *d* раз операции суммирования элемента *G*). Пусть тогда пара ключей первого абонента будет (d_A,Q_A), а пара второго абонента (d_B,Q_B). Перед исполнением протокола стороны должны обменяться открытыми ключами.

Первый абонент вычисляет $(x_k,y_k)=d_A \cdot Q_B$. Второй абонент вычисляет $(x_k,y_k)=d_B \cdot Q_A$. Общий секрет — x_k (x-координата получившейся точки). Большинство стандартных протоколов, базирующихся на ECDH, используют функции формирования ключа для получения симметричного ключа из значения x_k.

Вычисленные участниками значения равны, так как $d_A \cdot Q_B=d_A \cdot d_B \cdot G=d_B \cdot d_A \cdot G=d_B \cdot Q_A$. Из всей информации, связанной со своим закрытым ключом, первый абонент сообщает только свой открытый ключ. Таким образом никто кроме первого абонента не может определить ее закрытый ключ, кроме участника способного решить задачу дискретного логарифмирования на эллиптической кривой. Закрытый ключ Боба аналогично защищен. Никто кроме первого абонента или второго абонента не может вычислить их общий секрет, кроме участника способного разрешить проблему Диффи — Хеллмана.

Открытые ключи бывают либо статичными (и подтвержденные сертификатом) либо эфемерные (сокращённо ECDHE). Эфемерные ключи используются временно и не обязательно идентифицируют отправителя, таким образом, если требуется идентификация, подтверждение подлинности должно быть получено иным способом. Идентификация необходима для исключения возможности атаки посредника. Если первый абонент либо второй абонент используют статичный ключ, опасность атаки посредника исключается, но не может быть обеспечена ни прямая секретность, ни устойчивость к подмене при компрометации ключа, как и некоторые другие свойства устойчивости к атакам. Пользователи статических закрытых ключей вынуждены проверять чужой открытый ключ и использовать функцию формирования ключа на общий секрет чтобы предотвратить утечку информации о статично закрытом ключе. Для шифрования с другими свойствами часто используется протокол MQV.

При использовании общего секрета в качестве ключа зачастую желательно хешировать секрет чтобы избавиться от уязвимостей, возникающих после применения протокола.

MQV (**Менезес-Кью-Ванстоун**) — это аутентификационный протокол, базирующийся на алгоритме Диффи-Хеллмана. MQV предоставляет защиту против активных атак путем сочетания статического и временного ключей. Протокол может быть модифицирован для работы в произвольной конечной коммутативной группе, и, в частности, в группах эллиптических кривых, где известен как **ECMQV**.

MQV включен в проект по стандартизации криптосистем с открытым ключом — IEEE P1363.

Первый абонент имеет статическую ключевую пару (W_a,w_a), где W_a ее открытый ключ и w_a её секретный ключ. Второй абонент имеет статическую ключевую пару (W_b,w_b), где W_b его открытый ключ и w_b его секретный ключ. Определим R'. Пусть $R=(x,y)$ будет точкой на эллиптической кривой. Тогда $R'=(x \bmod 2^L)+2^L$, где $L=(\log_2 n+1)/2$ и n есть порядок используемого генератора точки P. Таким образом, R' есть первые L битов координаты x для R. Кроме того введем кофактор h, определенный как $h=|G|n$, где $|G|$ есть порядок группы G, причем следует учесть, что по техническим причинам должно выполняться требование: $gcd(n,h)=1$.

Пример.

Эллиптическая кривая E над полем $GF(2^{163})$ имеет порядок $2 \cdot P49$, где $P49$ — простое число состоящее из 49 цифр в десятичной записи.

$E:Y^2+XY=X^3+X^2+1$

Выберем неприводимый многочлен

$1+X+X^2+X^8+X^{163}$

И возьмём точку эллиптической кривой

P=(d42149e09429df4563ec1816488c92de89f93a9b2, ccd18d6cc3042c4c17a213506345c80965ac19476)≠0.

Проверим что ее порядок не равен 2

2P=(ccd18d6cc3042c4c17a213506345c809b5ac1d476, 835a2f56b88d6a249b4bd2a7550a4375e531d8a37).

Значит, её порядок равен порядку группы $2 \cdot P49$, а именно числу $P49$, и её можно использовать для построения ключа. Пусть $k_A=12$, $k_b=123$. Тогда открытые ключи участников протокола вычисляются как,

$k_A \cdot P=12 \cdot P=$(bd9776bbe87a8b1024be2e415952f527eee928b43, c67a28ed7b137e756c37654f186a71bf64e5ac546).

$k_B \cdot P=123 \cdot P=$(a5684e246044fc126e9832d17513387e474290547, 568b4137f09f5f79a8a6b0fe44cdf41d8e68ae2c6).

А общий секрет будет равен:

$k_B \cdot k_A \cdot P=k_A \cdot k_B \cdot P=12 \cdot 123 \cdot P=$(bb7856cece13c71919534878bcb6f3a887d613c92, f661ffdfe1ba8cb1b2ad17b6550c65aa6d4f07f41).

В качестве ключа симметричной системы используется значение (или его часть) x=bb7856cece13c71919534878bcb6f3a887d613c92.

Возможные перечни угроз безопасности и варианты их решения

№	Угроза	Варианты решения
1	Атака посредника	Использование статических ключей и идентификации пользователей (подтверждение пользователя)
2	Тайм атака	Использование алгоритма ecdsa25519
3	Атака 51%	При увеличении количества узлов - будет уменьшаться вероятность атаки, так как при создании узла программный комплекс должен подтвердить посредствам получения контрольной суммы файла реестра, его (реестра) реальное физическое локально существование
4	Взлом связки Public - Private keys	использование современного алгоритма формирование ключа на основе эллиптический кривых по алгоритму ecdsa25519
5	Атака окружения (легального) узла скомпрометированными узлами	Для минимизации вероятности осуществления подобной атаки используется алгоритм подтверждения легальности узла (предоставление контрольной суммы файла реестра, находящегося на локальном дисковом пространстве) совместно со случайным выбором возможного окружения, путем ранжирования узлов, содержащихся в списке и последующей их обработки.
6	Участие в раунде принятия решения по включению транзакцию по алгоритму византийских генералов скомпрометированного узла	Система не допускает до этапа принятия решения по транзакциям узлам, не способным подтвердить актуальность и реальное существование реестра и программного комплекса на локальном хранилище путем формирования контрольное суммы (алгоритм blake2s). Контрольная сумма (hash) проверяется в момент формирования белого списка узлов на следующий этап.
7	Кошелек слабо защищен от краж	Любая информация передаваемая в сети шифруется посредствам гомоморфного шифрования.
8	Атака Сибиллы	Узлы синхронизируют таблицу адресов узлов сети с остальными узлами и устанавливают соединения со случайным набором узлов, что

		позволяет избежать подмены пакетов и окружение легальных узлов
9	Проблема двойной траты	Минимизация вероятности выполнения подобной атаки путем сокращения времени на формирование пула транзакций, как результат сокращение времени на принятие решения; также при работе в многосерверном режиме обработки транзакций (параллельная обработка транзакций) используется синхронизация списков на главных узлах.
10	Перехват и изменение информации/транзакции при прохождении по сети	Использование новейших алгоритмов эллиптических кривых (ecdsa25519) для осуществления более криптоскойкой подписи и надежной защиты, для повышения надежности данных используется гомоморфный алгоритм шифрования, что позволяет увеличить скорость за счет обработки информации без необходимости полного ее декодирования
11	Вмешательство в работу с памятью	Использование языка разработки C++, что позволяет полностью контролировать память и при необходимости полностью управлять регистрами и проверять их содержимое с целью перехвата попытки компрометирования данных, находящихся на обработке в памяти
12	Изменение локального хранилища, с целью подмены легальных значений	Запись шифрованных данных и сжатие модифицированным алгоритмом deflate, что обеспечит повышенный уровень надежности хранения и более высокий коэффициент криптостойкости системы локального хранилища
13	Централизация обработки	В определенный математически промежуток (период) времени узел не может быть главным или доверенным дважды. Также данная концепция позволяет избежать окружения узла.

Смарт контракт

Смартконтракт в системе CREDITS это электронный алгоритм, описывающий набор условий, с помощью которого можно связывать действия и события в реальном мире или цифровых системах.

Для реализации самоуправляемых смартконтрактов требуется децентрализованная среда, полностью исключающая человеческий фактор, а для возможности использования в умном контракте передачи стоимости требуется независимая от центрального органа криптовалюта.

Сущности

Смартконтракт разрабатывается с использованием интерпретируемого языка JAVA, что дает возможность разрабатывать и проводить тестирование на абсолютно любой платформе без необходимости установки специальной среды разработки.

Для выполнения смартконтрактов используется виртуальная машина Java (далее JVM, поставляется по свободно распространяемой лицензии вместе с программным обеспечением платформы).

Смартконтракт в CREDITS состоит из следующих сущностей:

1. Свойство (public переменные) сущность системы, позволяющая хранить публичные

данные, необходимые для работы контракта в системе CREDITS.
2. Метод сущность системы CREDITS, отвечающая за соблюдение логики и

последовательности действий при проведении транзакции (действий по контракту).

Участники системы CREDITS подписывают смартконтракты, используя вызов методов, изменяющих свойства контракта,запуская процессы проверки выполнения условий проверки и согласования.
Смартконтракт после подписания сторонами вступает в силу. Для обеспечения автоматизированного исполнения обязательств непременно требуется среда существования, которая позволяет полностью автоматизировать выполнение пунктов контракта. Это означает, что смартконтракты смогут существовать только внутри среды, имеющей беспрепятственный доступ исполняемого кода к объектам смартконтракта.
Все условия контракта должны иметь математически обоснованную модель и описание и ясную логику исполнения. Таким образом, основной принцип смартконтракта состоит в полной автоматизации и достоверности исполнения договорных отношений между сторонами.

Метод смартконтракта

Методом смартконтрактов CREDITS называется сущность системы, отвечающая за соблюдение логики и последовательности действий при проведении транзакции (действий по контракту).

Логика и последовательность действий описывается программным кодом (модулем), содержащем команды, последовательное выполнение которых позволяет получить искомый результат. Данный код может оперировать системными командами (например, команда присваивания), пользовательскими командами (отдельно написанными функциями), свойствами контракта (статически или динамически проинициализированный переменные, доступные из любого метода контракта), а также методами из любого другого стороннего контракта, использование которого доступного не только владельцу подключаемого (стороннего) контракта. Для большей популяризации разработка ведется на JAVA.

Метод (программный код) допускает использование всех широко распространенных операторов (команд) скриптовых языков (операторы присваивания, условных и безусловных переходов), создание функций и процедур (подпрограмм), подключение сторонних библиотек.

Библиотеки для передачи в сеть транзакций, для использования на стороннем программном обеспечении

Для работы со сторонними сервисами с отправкой информации из сети о выполнении операции (транзакции) используется протокол HTTP/HTTPS (работа по 80/443 порту). Система использует стандартное определение протокола: заголовок (header) и тело запроса (body). Любой запрос отправляется по методу POST, что дает возможность повысить надежность передаваемых данных, чем при открытой передачи через метод GET.
Дополнительными условиями идентификации отправителя является указание в поле

```
Authorization = [ACCOUNT_HASH]
```

Тело запроса формируется в соответствии с выходными параметрами служб.
Для работы с пиринговой сетью CREDITS используются библиотеки разработанные на языках C++ или JAVA.
Функции, поддерживаемые библиотеками, делятся на четыри группы.

1. Общесистемные, обеспечивающие подключение и поддержку коннекта к сети, проверку работоспособности и заполнение полей в системных запросах, отправляемых в сеть, преобразование данных в соответствии с алгоритмами, необходимыми для системы
2. Работа с аккаунтом. Создание и разблокировка аккаунта, предоставление данных по аккаунту (баланс, кол-во транзакций и прочее)
3. Работа по формированию транзакций, отправляемых от имени аккаунта
4. Функции по работе с общедоступными данными. Запрос общих данных по транзакции, аккаунту