От нуля к Monero: первое издание

Техническое руководство по приватной цифровой валюте; для начинающих, любителей и экспертов

Опубликовано 26 июня 2018 (v1.0.0)

Kurt M. Alonso
kurt@oktav.se

KOE
ukoe@protonmail.com

(для вопросов и комментариев, пожалуйста, ставьте в копию обоих авторов)

Пожертвование позволяет нам поддерживать этот проект, поскольку Monero продолжает развиваться и расти (например, внедрение bulletproofs), а также для потготовки новых материалов и отчетов (например, Kovri). Ваша поддержка имеет большое значение!

Надеемся, вам понравится «От нуля к Monero»!

Monero (XMR) адрес для пожертвований:

43sHzpng70FAUMrRzg5RSg2X0YQbCSRYBRt4PV61ByqwY90vfRGqMenj3ZkEQEaXsf7edQtTitH5xKG3t27kkKafKX40FzY

Содержание

Глава 1

1 Введение

Цель блокчейнов состоит в обеспечении доверия между несвязанными сторонами без участия какой-либо доверенной третьей стороны.

Доверие достигается посредством использования криптографических артефактов, обеспечивающих данные, сохранённые в легкодоступной базе данных (блокчейне), виртуальной неизменяемостью и защищающих такие данные от фальсификации. Другими словами, блокчейн является публичной, распределённой базой данных, содержащей данные, правомерностькоторых не может быть оспорена какой-либо стороной.

Данные криптовалютных транзакций сохраняются в блокчейне, который служит публичным журналом (*леджером*) всех верифицированных валютных операций. Большая часть данных криптовалютных транзакций сохраняется в виде простого текста, что упрощает процесс верификации транзакций сообществом.

Очевидно, что открытый блокчейн не соответствует каким-либо принципам анонимности, так как он виртуально *придаёт огласке* полную историю транзакций всех транзакций, совершаемых его пользователями.

Для решения проблемы недостаточной анонимности пользователи таких криптовалют, как Bitcoin, могут «маскировать» свои транзакции, используя временные промежуточные адреса [24]. Тем не менее, обладая соответствующими средствами, можно проанализировать потоки и довольно часто связать истинных отправителей с получателями [33, 14, 28].

В отличие от таких валют Moneroпытается решить проблему анонимности путём сохранения в блокчейне только срытых одноразовых адресов получателей. При этом,подтверждение распределения средств в транзакции осуществляется при помощи кольцевых подписей. Применение этих методов не позволяет связать отправителей с получателями или отследить источник средств [4].

Помимо этого, суммы транзакций в блокчейне Monero скрыты криптографическими структурами, которые обеспечивают непрозрачность валютных потоков.

Результатом является высокий уровень анонимности криптовалюты.

1.1 Цели

Криптовалюта Monero появилась относительно недавно. Несмотря на это, наблюдается стабильный рост её популярности[1].К сожалению, документации, подробно описывающей механизмы, используемые этой валютой, практически нет. Хуже того, важные части теоретической концепции были опубликованы в работах, не прошедших независимой технической экспертизы. Такие работы нельзя считать полными. Помимо этого, в них содержатся ошибки. В большинстве случаев надёжным источником информации с точки зрения теоретической концепции Monero может служить только исходный код.

Мы намерены исправить эту ситуацию. Мы соберём глубокую информацию о внутренних принципах работы Monero, рассмотрим алгоритмы и криптографические схемы, а также обсудим достаточность уровня анонимности и безопасности пользователей, обеспечиваемого с их помощью.

За основу мы взяли комплект программного обеспечения Monero версии 0.11.1.0. Все механизмы проведения транзакций, описанные в данной работе, относятся именно к этой версии. Несмотря на то, что 0.12.0.0 является самой последней версией, в этой первой версии отчёта мы не стали рассматривать мультиподписи. Нами также никак не рассматривались схемы реализации опротестованных транзакций, даже несмотря на то, что они могут частично использоваться из соображений обратной совместимости.

1 На 28 декабря 2017 года Monero занимала 10 место по рыночной капитализации среди криптовалют (см. https://coinmarketcap.com/).

1.2 Читателю

Ожидается, что читатель владеет не только базовыми знаниями в области дискретной математики и алгебраических структур, но и, возможно, самыми общими представлениями в области криптографии. Также желательно, чтобы читатель имел общее представление о принципах работы таких криптовалют, как Bitcoin. Для технически ориентированных любителей нами в сносках нами была приведена информация, которая поможет восполнить возможные пробелы.

Обладающий такими знаниями читатель поймёт наше структурированное, последовательное описание составных элементов криптовалюты Monero.

Нами были намеренно пропущены или перенесены в сноски некоторые математические и технические детали, если они были необходимы для ясности. Нами также были пропущены конкретные подробности реализации в тех местах, где мы не сочли их важными. Наша цель состояла в том, чтобы представить материал на стыке математической криптографии и компьютерного программирования. При этом должны были сохраниться полнота информации и чёткость изложения концепции.

1.3 Истоки криптовалюты Monero

Криптовалюта Monero, которая изначально называлась BitMonero, была создана в апреле 2014 в качестве производной криптовалюты на базе протокола доказательстваконцепции CryptoNote.

CryptoNote является протоколом, разработанным самыми разными людьми. Первой значимой работой, в которой был описан этот протокол, стал документ, опубликованный в октябре 2013 [34]. Автор документа, который скрывался под псевдонимом Николас ванн Саберхаген (Nicolas van Saberhagen), предложил обеспечивать анонимность отправителей и получателей путём использования одноразовых адресов, а неотслеживаемость потоков — за счёт применения кольцевых подписей.

После этого анонимность была Monero усовершенствована за счёт реализации возможности сокрытия суммы, что описано Грэгом Максвеллом (Greg Maxwell) и другими авторами [23], а также улучшения кольцевых подписей Шеном Несером (Shen Noether) [27].

1.4 Структура документа

Как уже было сказано ранее, нашей целью является создание полного и последовательного описания криптовалюты Monero. Структура данного отчёта соответствует такой задаче.Читатель последовательно знакомится со всеми элементами, необходимыми для описания внутренних принципов работы валюты.

Для обеспечения полноты описания мы решили представить все базовые криптографические элементы, необходимые для понимания сложностей, связанных с Monero. Во второй главе нами рассматриваются важные аспекты криптографии на эллиптических кривых.

В третьей главе кратко описаны алгоритмы, связанные с кольцевыми подписями и используемые для обеспечения конфиденциальности транзакций и исключения возможности двойных трат.

В четвёртой главе нами рассматриваются криптографические механизмы сокрытия сумм.

Наконец, после описания всех компонентов, можно перейти и к схемам самих транзакций, и пятая глава посвящена таким схемам.

В приложениях A и B рассматриваются примеры структур транзакций, взятых из блокчейна. Так мы попытались связать теорию, изложенную в предыдущих частях работы, с её реализацией на практике.

Глава 2

2 Базовые концепции

2.1 Несколько слов о системе обозначений

Одна из центральных задач настоящего отчёта состояла в сборе, рассмотрении, корректировке и упорядочивании всей существующей информации, касающейся внутренних принципов работы криптовалюты Monero. И, в то же самое время, мы попытались указать все подробности, необходимые для конструктивного и последовательного представления материала.

Для выполнения этой задачи было необходимо установить ряд соответствующих обозначений. Среди прочего нами были использованы:

- символы нижнего регистра для обозначения простых значений, целых чисел, последовательностей, двоичных представлений и так далее;
- символывверхнего регистра для обозначения точек кривых и сложных структур.

В случае с символами с особым значением, мы старались использовать одни и те же обозначения во всём документе. Например, генератор кривых всегда обозначался нами как G, его порядок как l, приватные/публичные ключи по возможности обозначались как k/K, соответственно, и т.д.

Помимо этого, мы старались придать *концептуальности* нашему представлению алгоритмов и схем. Читатель, обладающий знаниями в области компьютерной науки, может почувствовать, что нами были опущены некоторые вопросы, например, связанные с двоичным представлением элементов, или же, в некоторых случаях, касающиеся способов реализации проведения конкретных операций.

Тем не мене, мы не считаем это упущением. Простые объекты, такие как целое число или последовательность, всегда могут быть представлены строкой бит. В случае с нашими алгоритмами такая вещь как *порядок байтов* (*endianness*) встречается довольно редко и в большинстве случаев является условной.

Точки эллиптической обычно обозначаются как пара (x, y) и, следовательно, могут быть представлены двумя целыми числами. Тем не мене, в мире криптографии практикуются методы *сжатия точек* (*point compression*), позволяющие представить точку, используя пространство только по одной координате. В случае с нашим концептуальным подходом использование или неиспользование методов сжатия точек часто имеет второстепенное значение. Тем не менее, по большей части, нами косвенно подразумевается их использование.

Нами также были свободно использованы хеш-функции безотносительно каких-либо конкретных алгоритмов. В случае с Monero это обычно вариант *Keccak*[2], однако, если такая информация не указывается, значит, это не так важно с точки зрения теории.[3]

Эти хеш-функции применимы к целым числам, последовательностям, точкам кривой или различным комбинациям этих объектов. Такие случаи следует рассматривать как хеши двоичных представлений или как совокупность таких представлений. В зависимости от контекста результат хеширования будет численным, будет представлен строкой бит или даже точкой кривой. Более подробно этот вопрос будет рассмотрен далее по мере необходимости.

---

2 Алгоритм хеширования Keccak лежит в основе стандарта Национального института стандартов и технологий США (NIST) *SHA-3*.

3 Хеш-функция берёт какое-либо сообщение **m** и возвращает хеш *h* (или *сжатую форму сообщения*) фиксированной длины, при этом, каждый возможный выход равновероятен заданному входу. Криптографические хеш-функции очень трудно обратить, у них есть интересная особенность, известная как *большой лавинный эффект* (*large avalanche effect*), когда очень похожие сообщения создают очень непохожие хеши, в результате чего трудно найти два сообщения с одной и той же сжатой формой.

2.2 Криптография на эллиптических кривых

2.2.1 Что такое эллиптические кривые

Конечное поле F_q, где q является простым числом более 3, это поле, сформированное последовательностью $\{0,1,2,\ldots,q-1\}$. Арифметические действия $(+,\cdot)$ и унарная операция $(-)$ являются вычисленным $(mod\, q)$.[4]

Обычно эллиптические кривые определяют как набор точек (x,y), соответствующих уравнению Вейерштрасса для заданной пары (a,b):

$$y^3 = x^3 + ax + b, \text{где } a,b,x,y \in F_q$$

Тем не менее, криптовалюта Monero использует специальную кривую, известную тем, что она обеспечивает более высокий уровень безопасности, чем другие обычно используемые *NIST* кривые, а также криптографические примитивы, демонстрирующие превосходные показатели. Эта кривая относится к категории так называемых *скрученных кривых Эдвардса* (*Twisted Edwards curves*), которые обычно представлены следующим выражением:

$$ax^2 + y^2 = 1 + dx^2 y^2, \text{ где } a,d,x,y \in F_q$$

Из чего следует, что нам лучше использовать вторую форму. Её преимуществом, помимо использования уже упомянутого уравнения Вейерштрасса, является то, что её криптографические примитивы требуют меньшего количества арифметических действий, что делает работу криптографических алгоритмов быстрее. Подробная информация содержится в работе [12] Бернштейна и др. (*Bernstein et al.*).

Допустим, что $P_1 = (x_1, y_1)$ и $P_2 = (x_2, y_2)$ являются двумя точками, принадлежащими скрученной кривой Эдврадса (далее просто именуемой EC). После этого можно перейти к определению операции сложения[5]:

---

4 «Вычисленный $(mod\, q)$» означает $(mod\, q)$, который производится для каждого вида арифметического действия между двумя элементами поля или дл отрицания отдельно взятого элемента поля. Например, при заданном простом поле F_p, где $p = 29$, $17 + 20 = 8$, так как $37\,(mod\, 29) = 8$. Точно так же $-13 = (-13)(mod\, 29) = 16$.

В данном случае модель (положительный) для $a\,(mod\, b) = c$ определяется как $a = bx + c$, где $0 \le c < b$ и x являются целым числом со знаком, которое отбрасывается. Представьте числовую ось. Встаньте в точку a. Двигайтесь к нулю с шагом = b, пока не достигните числа ≥ 0 и ¿ b. Это и будет c.

5 $x_3 = (1 + dx_1 x_2 y_1 y_2)^{-1}(x_1 y_2 + x_1 y_2)(mod\, q)$можно вычислить, используя свойства модульного умножения и сложения $(A \circ B)(mod\, C) = [A(mod\, C)] \circ [B(mod\, C)](mod\, C)$, а также модульную мультипликативную инверсию. Начинать следует со скобок.

Модульная мультипликативная инверсия определяется как целое числоx таки образом, что для $x = a^{-1}(mod\, n)$, $ax \equiv 1(mod\, n)$ для $0 \le x < n$ и дляa и n является относительно простой. Расширенный алгоритм Евклида позволяет найти x следующим образом:

$Q = 0;\ newQ = 1;\ R = n;\ newR = a$
при этом $newR \ne 0$
$quotient = integer(R / newR)$
$(Q / newQ) = (newQ, Q - quotient * newQ)$
$(R / newR) = (newR, R - quotient * newR)$
если $R \le 1$, то (при$Q < 0\ return\, Q + n$ ещё $return\, Q$) иное решение отсутствует.

Следует обратить внимание на *согласованность* членов уравнения: в уравнении $a \equiv b(mod\, c)\, a$ согласуется с $b(mod\, c)$, что означает, что $a(mod\, c) = b(mod\, c)$.

$$x_3 = \frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2} (mod\, q)$$

$$y_3 = \frac{y_1 y_2 + a x_1 x_2}{1 - d x_1 x_2 y_1 y_2} (mod\, q)$$

Эти формулы добавления также применяются для дублирования точек, то есть, для случаев, когда необходимо получить $P_1 = P_2$. Чтобы выделить точку, необходимо инвертировать её координаты $(x, y) \to (x, -y)$и добавить точку. При каждом появлении «отрицательного» элемента $-x$ поляF_q в этом отчёте, фактически это будет $-x\,(mod\, q)$.

В результате использования описанной операции добавления эллиптические кривые приобретают структуру *абелевой группы*[6]. Всякий раз при использовании этой операции P_3 является точкой на «оригинальной» эллиптической кривой, или, другими словами, $x_3 y_3 \in F_q$.

Каждая точка P кривой EC может формировать подгруппу порядка (размера) u из нескольких других точек EC, используя свои кратные. Например, подгруппа некоторой точки P может быть порядка 5 и включать в себя другие точки $(0P, P, 2P, 3P, 4P)$, каждая из которых будет находиться на EC. В точке$5P$появляется так называемая *бесконечно удалённая точка*, которая, вероятнее всего, будет находиться в нулевой точке EC с координатами (0, 1).

Это очень удобно: $0P = 5P$ и $5P + P = P$. Это указывает на цикличность подгруппы[7]. Все точки P кривой EC формируют циклические подгруппы. Если точка P формирует подгруппу, порядок[8] которой являетсяпростым, то все входящие в эту подгруппу точки (за исключением бесконечно удалённой точки) также формируют такую подгруппу.

Каждая кривая EC имеет порядок N, равный общему количеству точек кривой, включая бесконечно удалённую точку, а порядки всех подгрупп, сформированных точками, являются делителями N (по *теореме Лагранжа*).

У кривых EC, выбранных для криптографии, обычно $N = hl$, где l является достаточно большим простым числом (как 160 бит). Одна точка из подгруппы с размером l (следует отметить, что h называется кофактором) выбирается в качестве генератора и обозначается как G. Для каждой другой точки P в подгруппесуществует целое число n, соответствующее $P = nG$.[9]

---

6 Точное определение этого понятия можно прочитать по ссылке https://brilliant.org/wiki/abelian-group/.

7 Цикличность подгруппы означает, что для подгруппы точки P порядка u и любого целого числа n $nP = [n\,(mod\, u)]P$.

8 Чтобы определить порядок u подгруппы точки P, необходимо:
1 найти N (например, используя *алгоритм Шуфа*);
2 найти все делители N;
3 для каждого делителя nуN найти nP;
4 самый малый делительn при котором $nP = 0$ будет порядком u подгруппы.

9 Допустим, что у нас есть точка P порядка N ($N = hl$). Любая другая точка кривой EC может быть найдена как $P_i = n_i P'$. Если $P_1 = n_1 P'$ имеет порядок l, то любая точка $P_2 = n_2 P'$ порядка l должна находиться в той же подгруппе, что и P_1, поскольку $lP_1 = 0 = lP_2$, и если $l(n_1 P') \equiv l(n_2 P') \equiv NP = 0$, то n_1 и n_2 будут кратными h. Другими словами, подгруппа, сформированная кратными (hP'), всегда будет включать в себя точки P_1 и P_2. Более того, $h(n'P') = 0$, если n' будет кратным l, и таким образом, n' умноженное на P' может дать только h точек перед $n' = hl$, которое по циклу возвращается обратно к 0: $hlP' = 0P' = 0$. Таким образом, на кривой EC, где hP будет равным 0, количество точек будет составлять только h.
Чтобы найти подходящее значение G, необходимо:
1 найти N кривой EC, выбрать подгруппу порядка l, вычислить $h = N/l$;
2 выбрать произвольную точку P на кривой EC;
3 вычислить $G = hP$;
4 Если $G = 0$, вернуться к шагу 3, G формирует подгруппу порядка l.

Вычисление скалярного произведения nP не представляет сложности[10], в то время как нахождение такого n, чтобы $P_1 = nP_2$ является сложным с точки зрения вычисления. По аналогии с модульной арифметикой, эту проблему часто называют *проблемой дискретного логарифмирования* (*discrete logarithm problem*, *DLP*). Другими словами, скалярное умножение может рассматриваться в качестве необратимой функции, что позволяет использовать эллиптические кривые в криптографии.

2.2.2 Использование криптографии на эллиптических кривых для создания публичных ключей

Криптографические алгоритмы создания публичных ключей могут создаваться путём аналогичным модульной арифметике.

Допустим, k является произвольно выбранным числом, соответствующим условию $1 < k < l$, и мы назовём его *приватным ключом*. Необходимо вычислить соответствующий *публичный ключ* $K = kG$.

Из-за *проблемы дискретного логарифмирования* (DLP) мы не можем просто вывести k только на основе K. Это свойство позволяет нам использовать значения (k, K) в криптографических алгоритмах создания общего публичного ключа.

2.2.3 Протокол обмена ключами Диффи-Хеллмана на эллиптических кривых

Базовый обмен секретными ключами Диффи-Хеллмана (*Diffie-Hellman*) между *Элис* и *Бобом* мог бы происходит следующим образом.

1. Элис и Боб генерируют собственные приватные/публичные ключи (k_A, K_A) и (k_B, K_B). Оба публикуют или обмениваются своими публичными ключами, а приватные ключи оставляют у себя.
2. Очевидно, что

$$S = k_A K_B = k_A k_B G = k_B k_A G = k_B K_A.$$

Элис может приватно вычислить $S = k_A K_B$, а Боб может вычислить $S = k_B K_A$. Это позволяет им использовать это единственное значение в качестве общего секрета.

Сторонний наблюдатель не сможет вычислить общее секретное значение из-за проблемы DLP, не позволяющей найти k_A или k_B.

2.2.4 Подписи DSA на эллиптических кривых (ECDSA)[11]

---

10 Скалярное произведение nP эквивалентно $(((P+P)+P)\dots)$. Ещё один базовый, но не мене мощный алгоритм, позволяющий сократить вычисление nP, известен как *алгоритм удвоения и добавления* (*double-and-add*). Мы продемонстрируем его на примере. Допустим, $n = 7$, таким образом, $nP = P + P + P + P + P + P + P$. Теперь разобьём точки на группы по две. $(P+P)+(P+P)+(P+P)+P$. И ещё раз на группы по две. $[(P+P)+(P+P)]+(P+P)+P$. Общество количество действий сложения точек составляет от 6 до 4, поскольку $(P+P)$ требуется вычислить только один раз.

Первым шагом реализации алгоритма удвоения и добавления является преобразование n в двоичную форму, после чего происходит циклическое повторение через полученную последовательность для получения суммы $Q = nP$. Необходимо помнить о необходимости использования операции сложения точек, о которой говорилось в Разделе 2.2.1. Этот алгоритм предполагает значительный порядок байтов

$n_{scalar} \rightarrow n_{binary}$; $A = [n_{binary}]$; $Q = 0$, бесконечно удалённая точка, $R = P$

для $k = (A_{size} - 1)\dots 0$

  если $A[k] = ¿\,1$

    $Q + ¿\,R$

  $R + ¿\,R$

результат Q

11 См. работу [18] и ANSIX9.62. В этих документах содержится подробная информация, которая не приводится в данном отчёте.

Обычно криптографическая подпись строится на криптографическом хеше сообщения, а не на самом сообщении[12]. Тем не менее. В этом отчёте нами будет свободно использоваться термин *сообщение* для определения сообщения и/или его значения хеша.

Подпись

Предположим, что у Элис есть пара, состоящая из приватного/публичного ключей, (k, K). Чтобы уникально подписать произвольное сообщение *m*, она могла бы сделать следующее [17].

1. Вычислить хеш сообщения, используя криптографически безопасную хеш-функцию $h = H(m)$.
2. Допустим, что h' является самыми левыми (в большом порядке байтов) битами L_l функции *h*, где L_l имеет длину *l* в битах.
3. Сгенерировать произвольное целое число *r*, таким образом, чтобы $1 < r < l$ и вычислить $P = (x, y) = rG$.
   Если $x' = x \pmod l = 0$, необходимо сгенерировать другое произвольное целое число.
4. Вычислить $s = r^{-1}(h' + x'k) \pmod l$.[13] Если $s = 0$, необходимо вернуться к предыдущему шагу и повторить его.
5. Подпись будет следующей: (x', s).

Верификация

Любая третья сторона, которой известны параметры области определения *D* кривой EC, подпись (x', s) и метод подписания, *m* и хеш-функция, а также *K*, может верифицировать подпись[14], то есть, доказать, что *s* была создана владельцем *k* для сообщения *m* путём следующих вычислений:

проверить, что x' и s находятся в пределах интервала $[1, q-1]$

$$u_1 = s^{-1} h' \pmod l$$

$$u_2 = s^{-1} x' \pmod l$$

$Q = u_1 G + u_2 K$; если $Q = 0$, отклонить подпись.

Подпись будет действительна в том случае и только в том случае, если первая координата $Q = (x_Q, y_Q)$ будет соответствовать

$$x_Q \equiv x' \pmod l$$

Почему это работает

Причиной является тот факт, что

$$Q = u_1 G + u_2 K$$

$$¿\ s^{-1} h' G + s^{-1} x' kG$$

$$¿\ s^{-1}(h + x'k) G$$

12 Подписание хеша сообщения, а не самого сообщения, облегчает процесс при наличии сообщений различного размера.

13 Для этого вычисления необходимо обратиться к сноске 5.

14 В работе по ECDSA [18] рекомендуется производить валидацию действительности D и K перед тем, как верифицировать подпись. D рассматривается в работе [32].

Так как $s = r^{-1}(h' + x'k) \pmod{l}$, следовательно[15], $r \equiv s^{-1}(h' + x'k) \pmod{l}$, поэтому[16]

$$Q = rG$$

Следовательно, владелец k создал s для m, то есть, он подписал сообщение.

2.3 Кривая Ed25519

Для реализации криптографических операций Monero использует определённую скрученную эллиптическую кривую Эдвардса, *Ed25*519, бирациональный эквивалент[17] кривой Монтгомери *Curve25519*.

Как кривая Ed25519, так и кривая Curve25519 были описаны в работах Бернштейна и др. (*Bernstein et al.*) [10, 11, 13].

Кривая определяется через простое поле $F_{2^{255}-19}$ при помощи следующего уравнения:

$$-x^2 + y^2 = 1 - \frac{121665}{121666} x^2 y^2$$

Использование этой кривой решает ряд вопросов, поднятых криптографическим сообществом. Прекрасно известно, что стандартные алгоритмы NIST[18] не лишены недостатков. Например, недавно стало очевидно, что алгоритм генерации случайных чисел *PNRG* имеет определённый изъян и может содержать потенциальную лазейку [16]. Если смотреть с более широкой перспективы, кривые, имеющие отношение к NIST, также имеют отношение и к Агентству национальной безопасности (NSA), на что криптографическое сообщество смотрит с определённым подозрением. NSA печально известно тем, что использует свою власть над NIST с целью ослабления криптографических алгоритмов [6].

Кривая Ed25519 не запатентована (в работе [20] обсуждается этот вопрос), и команда, стоящая за её созданием, занималась разработкой и адаптацией базовых криптографических алгоритмов, ориентируясь на их эффективность [13]. Но что более важно, в настоящее время кривая считается безопасной.

Скрученные эллиптические кривые Эдвардса имеют порядок, который можно выразить как $N = 2^c l$, где l является простым числом, а c положительным целым числом. В случае с кривой Ed25519 порядок составляет десятичным числом 76:
$2^3 \cdot 7237005577332262213973186563042994240857116359379907606001950938285454250989$

2.3.1 Двоичное представление

Элементы $F_{2^{255}-19}$ являются 256-битными целыми числами. Другими словами, они могут быть представлены 32 байтами. Так как на каждый элемент требуется только 255 бит, самым значимым битом всегда является нулевой.

Следовательно, любая точка кривой Ed25519 может быть выражена 64 байтами. Тем не менее, методы *сжатия точек*, описанные ниже. Позволяют снизить это количество вдвое до 32 байтов.

2.3.2 Сжатие точек

Кривая Ed25519 обладает свойством, которое заключается в том, что её точки можно легко сжать настолько, что представление одной точки займёт пространство всего одной координаты.

---

15 Правило модельной мультипликативной инверсии гласит:
Если $ax \equiv b \pmod{n}$, *где* a *и* n *являются относительно простыми, решение этого линейного сравнения производится как* $x = a^{-1} b \pmod{n}$.[5]
Следовательно, так как l является простым элементом, $s = r^{-1} z \pmod{l} \rightarrow r \equiv s^{-1} z \pmod{l}$, где $z = (h' + x'k)$.

16 Возвращаясь к сноске 7: $nP = n \pmod{u} P$

17 Если не вдаваться в подробности, то бирациональным эквивалентом можно назвать изоморфизм, который можно выразить в рациональных терминах.

18 Национальный института стандартов и технологий США, https://www.nist.gov/

Мы не станем углубляться в математические подробности, необходимы для того, чтобы объяснить это, но мы можем кратко показать, как это работает [11].

Данная схема сжатия точек основана на преобразовании уравнения скрученной кривой Эдвардса: $x^2=(1-y^2)/(a-dy^2)$, которое демонстрирует, что у x для каждого y может быть только два возможных значения (+ или –). Элементы поля x и y вычисляются $(mod\, q)$, поэтому фактические отрицательные значения отсутствуют. Тем не менее, $(mod\, q)$ от $-x$ изменит значение между нечётными и чётными, так как q является нечётным. Например, $-3(mod\, 5)=2$, $-6(mod\, 9)=3$. Другими словами, элементам поля x и $-x$ присваиваются различные чётные/нечётные значения.

Если нам известно, что x имеет чётное значение, но при заданном значении y преобразованное уравнение кривой даёт нечётное число («положительный» x), то мы знаем отрицание, при котором значение даст нам правильный x. Эта информация может содержаться в одном бите, а для y есть запасной бит, что очень удобно.

Предположим, что нам нужно сжать точку (x, y). Мы используем короткое представление целых чисел. Следует отметить, что $q=2^{255}-19$ согласуется с $5\,(mod\, 8)$.

Шифровка

Мы присваиваем наиболее значимому биту y нулевое значение, если x имеет чётное значение, и 1, если нечётное. Полученное значение y' будет представлять точку кривой.

Дешифровка

Берём сжатую точку y', а затем копируем её наиболее значимый бит в контрольный бит чётности b, перед тем как присвоить ему нулевое значение. Это будет y.

Допустим, $u=y^2-1$, а $v=dy^2+1$.

Вычисляем[19] $x=uv^3(uv^7)^{(q-5/8)}$

1. Если $ux^2=u(mod\, q)$, то оставляем x.
2. Далее задать $x=x\,2^{(q-1)/4}(mod\, q)$.
3. Используя контрольный бит чётности b, полученный при выполнении прошлого этапа, если $b\neq$ наименее значимому биту x, получаем $q-x$ (то же, что и $-x(mod\, q)$), в противном случае получаем x.

2.3.3 Алгоритм подписи EdDSA

Бернштейном и его командой был разработан ряд базовых алгоритмов, основанных на кривой Ed25519.[20] Для примера нами описана предельно оптимизированная и безопасная альтернатива схеме подписи ECDSA, которая, со слов её авторов, позволяет создавать более

---

19 Как и в случае с алгоритмом удвоения и добавления, описанным в сноске 10, мы можем найти $a^e(mod\, n)$ следующим образом:

$e_{scalar} \rightarrow e_{binary}$; $A=[e_{binary}]$; $Q=1$, $R=a$

для $k=$¿

если $A[k]=$¿ 1

$Q=Q*R(mod\, n)$

$R=R*R(mod\, n)$

результат Q

Необходимо отметить, что модульное скалярное умножение может быть произведено при помощи алгоритма, описанного в сноске 10 путём замены добавления точки кривой EC модульным добавлением.

Это также обеспечивает наглядную, но более медленную альтернативу расширенному алгоритму Евклида для модульной мультипликативной инверсии (см. сноску 5). Если n является простым числом, мы можем использовать малую теорему Ферма

$$a^{-1}\equiv 1(mod\, n) \quad a^{-1}\equiv a^{n-2}(mod\, n)$$

20 В работе [13] описана группа эффективных операций со скрученной эллиптической кривой Эдвардса (то есть, добавление точки, удвоение, смешанное добавление и т.д.)

100 000 подписей в секунду, используя обычный потребительский процессор Intel Xeon [11]. Алгоритм также описан в Internet RFC8032 [19].

Помимо прочего, вместо генерации случайных целых чисел, в данном случае используется значение хеша, выведенное на основе приватного ключа отправителя и на основе самого сообщения. Это позволяет обойти все уязвимости, связанные с реализацией генераторов случайных чисел. Также, другая цель алгоритма состоит в том, чтобы не допустить несанкционированного доступа к конфиденциальным данным или непрогнозируемым областям памяти, что позволяет не избежать так называемых *атак кэша по времени*.

В этой работе исключительно для наглядности нами кратко описаны этапы алгоритма. Полное описание и пример реализации на языке Python можно найти в работе [19].

Подпись

1. Допустим, h_k является хешем $H(k)$ приватного ключа k подписавшегося. Вычисляем $r=H(h_k,m)$ хешированного приватного ключа и сообщения.
2. Вычисляем $R=rG$ и $s=(r+H(R,K,m)\cdot k)$.
3. Подпись будет представлена парой (R,s).

Верификация

Верификация производится следующим образом.

1. Вычисляем $h=H(R,K,m)$.
2. Если равенство $2^c sG=2^c R+2^c hK$ соблюдается, значит подпись является действительной.[21]

Почему это работает

$$2^c sG=2^c\left((r+H(R,K,m)\cdot k)\cdot G=2^c R+2^c H(R,K,m)\cdot K\right)$$

Двоичное представление

По умолчанию подписи EdDSA для представления требуется 64+32 байта. Тем не менее, RFC8032 предполагает, что точка R является сжатой, что снижает требования к занимаемому пространству до 32+32 байт.

21 Член уравнения 2^c взят из общей формы Бернштейна алгоритма EdSDA [13]. Согласно этой работе, хотя этого и не требуется для адекватной верификации, удаление 2^c усиливает уравнение.

Глава 3

3 Кольцевые подписи

Кольцевые подписи состоят из кольца и подписи. Каждая *подпись* генерируется при помощи одного приватного ключа и ряда несвязанных между собой публичных ключей. Каждое *кольцо* является набором публичных ключей, в котором есть и публичный ключ приватного ключа, а также набором несвязанных между собой публичных ключей. Кто-то, проводящий верификацию подписи, не сможет сказать, кто из членов кольца соответствует тому приватному ключу, который создал его.

Кольцевые подписи изначально назывались *групповыми подписями* (*Group Signatures*), поскольку считались способом доказательства того, что подписант принадлежит к группе, без какой-либо идентификации такого подписанта. В контексте транзакций Monero, подписи помогают сделать потоки валюты неотслеживаемыми, не создавая вектора атаки на денежные ресурсы, находящиеся в обращении.

Схемы организации кольцевых подписей обладают рядом свойств, которые способствуют обеспечению конфиденциальности транзакций.

- **Анонимность подписанта**. Наблюдатель должен быть в состоянии определить, что подписант является членом кольца, но недолжен знать, каким именно.[22] Monero использует это для маскировки источника средств при проведении каждой транзакции.
- **Связываемость**. Если приватный ключ используется для того, чтобы подписать два различных сообщения, то эти сообщения становятся связанными.[23] Как будет показано далее, это свойство помогает предотвратить атаки, связанные с двойной тратой Monero.
- **Невозможность подделки**. Никакой злоумышленник не сможет сгенерировать фальшивую подпись.[24] Это свойство обеспечивает защиту Monero от подделки.

3.1 Связываемые спонтанные анонимные групповые подписи (LSAG)

Изначально (см. Чаум (*Chaum*) [15]), схемы групповых подписей требовали настройки системы, а в некоторых случаях и управления доверенным лицом с целью предотвращения использования незаконных подписей и, в случае с некоторыми схемами, разрешения спорных ситуаций. Использование группового секрета не желательно, так как создается риск раскрытия, что ставит под угрозу анонимность. Кроме того, необходимость в координации между членами группы (то есть в настройке и управлении) не масштабируется, независимо от того, происходит это в пределах небольших групп или целого сообщества.

Лю и др. (*Liu et al.*) в работе [21] была представлена более интересная схема, основанная на исследованиях Ривеста и др. (*Rivest et al.*) [31]. Авторами был подробно изложен алгоритм формирования групповых подписей, характеризуемый тремя свойствами: анонимность, связываемость и спонтанность. Другими словами, владелец приватного ключа мог

22 Анонимность в случае совершения какого либо действия с точки зрения «анонимного ряда», который включает в себя «всех людей, которые могли совершить такое действие». Самым большим анонимным рядом является «человечество», а в случае с Monero это *уровень смешивания v*. Термин «смешивание» подразумевает количество ложных членов в каждой кольцевой подписи. Если значение смешивания составляет $v = 4$, значит присутствует 5 возможных подписантов. Расширение ряда анонимности усложняет процесс определения реальных участников транзакции.

23 Свойство связываемости не имеет отношения к публичным ключам, не используемым для подписания. То есть, участник кольца, чей публичный ключ был смешан в других подписях, не будет связан.

24 Определенные схемы кольцевых подписей, включая ту, которая используется Monero, довольно устойчивы к атакам, осуществляемой путём адаптивной выборки сообщений и адаптивной выборки публичного ключа. Злоумышленник, который может получить действительные подписи для сообщений и соответствующие публичные ключи в случае с такими кольцами, не сможет определить, как сформировать подпись даже для одного сообщения. Это называется экзистенциальной невозможностью подделки, см. [27] и [21].

сгенерировать одну анонимную подпись, выбрав любой ряд сопоподписантов из списка возможных приватных ключей, не сотрудничая при этом с кем-либо.[25]

Подпись

Допустим, m является подписываемым сообщением, $R=\{K_1, K_2, \ldots, K_n\}$ является набором определённых публичных ключей (группой/кольцом), а k_π является приватным ключом подписанта, соответствующим его публичному ключу[26] $K_\pi \in R$, где π является секретным индексом. Допустим наличие двух хеш-функций H_n и H_p, соответствующих числам от 1 до l,[27] и точкам кривой EC[28,29].

1. Вычислить образ ключа $\widetilde{K}=k_\pi H_p(R)$.
2. Сгенерировать случайное число[30] $\alpha \in_R Z_l$ и случайные числа $r_i \in_R Z_l$ для $i \in \{0,1,\ldots,n\}$ за исключением $i=\pi$.
3. Вычислить

$$c_{\pi+1}=H_n\left(R, \widetilde{K}, m, \alpha G, \alpha H_p(R)\right)$$

4. Для $i=\pi+1, \pi+2, \ldots, n, 1, 2, \ldots, \pi-1$, заменив $n+1 \to 1$, вычислить

$$c_{i+1}=H_n\left(R, \widetilde{K}, m, \left[r_i G+c_i K_i\right], \left[r_i H_p(R)+c_i \widetilde{K}\right]\right)$$

5. Найти такое значение r_π, чтобы $\alpha=r_\pi+c_\pi k_\pi \pmod{l}$.

Кольцевая подпись содержит подпись $\sigma(m)=\left(c_1, r_1, r_2, \ldots, r_n, \widetilde{K}\right)$ и кольца R.

Верификация

Процесс верификации означает доказательство того, что $\sigma(m)$ является действительной подписью, созданной с использованием приватного ключа, соответствующего публичному ключу в кольце R, и производится следующим образом.

1. Для $i=1, 2, \ldots, n$, заменив $n+1 \to 1$, итеративно вычислить

$$z_i^{'}=r_i G+c_i K_i$$
$$z_i^{''}=r_i H_p(R)+c_i \widetilde{K}$$
$$c_{i+1}^{'}=H_n\left(R, \widetilde{K}, m, z_i^{'}, z_i^{''}\right)$$

2. Если $c_1^{'}=c_1$, то подпись является действительной. Следует отметить, что $c_1^{'}$ является в последнюю очередь.

Почему это работает

Мы можем убедиться в том, что алгоритм работает, рассмотрев следующий пример. Возьмём кольцо $R=\{K_1, K_2, K_3\}$, в котором $k_\pi=k_2$. Сначала идёт подпись.

---

25 В рамках схемы LSAG связываемость реализуется только в случае с теми подписями, которые используют кольца с теми же участниками и в том же самом порядке, то есть, «абсолютно то же самое кольцо». Фактически это «одна анонимная подпись на кольцо». Связанные подписи могут быть присоединены к различным сообщениям.

26 Примечание: $K_\pi \in R$ означает, что K_π является членом ряда R.

27 В случае Monero хеш-функция $H_n(x)$ = sc_reduce32(*Keccack*(*x*)), где *Keccack* является основой SHA3, а sc_reduce32() определяет 256-битный результат в пределах от 1 до l.

28 Совершенно неважно, буду точки H_p сжатыми или нет. Они всегда могут быть развёрнуты.

29 Monero использует хеш-функцию, которая выдаёт точки кривой напрямую, а не путём вычисления какого-либо целого числа, которое затем умножается на G. H_p была бы разбита, если бы кто-то нашёл способ найти промежуточный итог n_x. $n_x G=H_p(x)$

30 Примечание: $\alpha \in_R Z_l$ означает, что α случайно выбирается из $\{1,2,\ldots,l\}$.

1. Вычислить образ ключа $\widetilde{K}=k_\pi H_p(R)$.
2. Сгенерировать случайные числа α, r_1, r_3.
3. Произвести шифрование:

$$c_3=H_n\left(\ldots,\alpha G,\alpha H_p(R)\right)$$

4. Произвести итерацию:

$$c_1=H_n\left(\ldots,\left[r_3G+c_3K_3\right],\left[r_3H_p(R)+c_3\widetilde{K}\right]\right)$$
$$c_2=H_n\left(\ldots,\left[r_1G+c_1K_1\right],\left[r_1H_p(R)+c_1\widetilde{K}\right]\right)$$

5. Замыкаем цикл: $r_2=\alpha-c_2k_2 \pmod l$

Мы можем убедиться заменить α на c_3, чтобы понять, откуда взялось слово «кольцо».

$$c_3=H_n\left(\ldots,\left[(r_2+c_2k_2)G\right],\left[(r_2+c_2k_2)H_p(R)\right]\right)$$
$$c_3=H_n\left(\ldots,\left[r_2G+c_2K_2\right],\left[r_2H_p(R)+c_2\widetilde{K}\right]\right)$$

Затем производится верификация с использованием R и $\sigma(m)=\left(c_1,r_1,r_2,r_3,\widetilde{K}\right)$.

1. r_1 $\quad z_1'=r_1G+c_1K_1 \quad z_1''=r_1H_p(R)+c_1\widetilde{K}$
$$c_2'=H_n\left(\ldots,\left[r_1G+c_1K_1\right],\left[r_1H_p(R)+c_1\widetilde{K}\right]\right)$$

2. r_2 $\quad z_2'=r_2G+c_2K_2 \quad z_2''=r_2H_p(R)+c_2\widetilde{K}$
$$c_3'=H_n\left(\ldots,\left[r_2G+c_2K_2\right],\left[r_2H_p(R)+c_2\widetilde{K}\right]\right)$$

3. r_3 $\quad z_3'=r_3G+c_3K_3 \quad z_3''=r_3H_p(R)+c_3\widetilde{K}$
$$c_1'=H_n\left(\ldots,\left[r_3G+c_3K_3\right],\left[r_3H_p(R)+c_3\widetilde{K}\right]\right)$$

таким образом, получаем $c_1'=c_1$.

Связываемость

Дано: фиксированный ряд публичных ключей R и две действительные подписи для различных сообщений,

$$\sigma=\left(c_1,s_1,\ldots,s_n,\widetilde{K}\right)$$
$$\sigma'=\left(c_1',s_1',\ldots,s_n',\widetilde{K}'\right)$$

если $\widetilde{K}=\widetilde{K}'$, то, очевидно, обе подписи принадлежат к одному кольцу и приватному ключу, так как $\widetilde{K}=k_\pi H_p(R)$.

Несмотря на то, что наблюдатель может связать σ и σ', но не сможет узнать, какой K_i в R является действующим, не решив DLP или не проанализировав R каким-либо образом (как, например, не узнав все k_i при $i\neq\pi$, или же вычислив k_π).[31]

31 LSAG не поддаётся поделке, поэтому никакой злоумышленник не сможет сгенерировать действительную кольцевую подпись, не зная приватного ключа. Если он придумает фальшивый образ $\widetilde{K}$ и запустит вычисление подписи, используя $c_{\pi+1}$, то, не зная k_π, он не сможет вычислить число $r_\pi=\alpha-c_\pi k_\pi$, которое позволит получить $\left[r_\pi G+c_\pi K_\pi\right]=\alpha G$. Верификатор отклонит его подпись. В работе Лю и др. доказано, что создание подделок, которые могли бы пройти верификацию, крайне маловероятно [21].

3.2 Обратные связываемые спонтанные анонимные групповые подписи (bLSAG)

В схеме LSAG подписей связываемость подписей при помощи одного и того же приватного ключа может быть гарантирована только в том случае, если кольцо является постоянным. Это очевидно следует из определения $\widetilde{K}=k_\pi H_p(R)$.

В данном разделе нами будет представлена улучшенная версия алгоритма LSAG, в которой связываемость не зависит от соподписантов кольца.

Данная модификация алгоритма рассматривается в работе [26] и основана на публикации А. Бэка (*A. Back*) [9], касающейся алгоритма кольцевых подписей CryptoNote [34].

1. Вычислить образ ключа $\widetilde{K}=k_\pi H_p(K_\pi)$.
2. Сгенерировать случайное число $\alpha \in_R Z_l$ и случайные числа $r_i \in_R Z_l$ для $i \in \{0,1,\ldots,n\}$ за исключением $i=\pi$.
3. Вычислить
   $c_{\pi+1}=H_n(m,\alpha G,\alpha H_p(K_\pi))$
4. Для $i=\pi+1,\pi+2,\ldots,n,1,2,\ldots,\pi-1$, заменив $n+1 \to 1$, вычислить
   $c_{i+1}=H_n(m,[r_i G+c_i K_i],[r_i H_p(K_i)+c_i \widetilde{K}])$
5. Найти значение $r_\pi=\alpha-k_\pi c_\pi \pmod l$.

Подпись будет следующей: $\sigma(m)=(c_1,r_1,\ldots,r_n,\widetilde{K})$.

Как и в случае с оригинальной схемой LSAG, верификация происходит путём перерасчёта значения c_1.

Правильность также может быть продемонстрирована (то есть, «как это работает») подобно тому, как это делается при использовании схемы LSAG.

Внимательный читатель, вне всякого сомнения, заметит, что образ ключа $\widetilde{K}$ зависит только от ключей истинного подписанта. Другими словами, две подписи теперь можно будет связать, если и только если для создания подписи использовался один и тот же приватный ключ. При реализации схемы bLSAG кольца используются только для сокрытия личности каждого из подписантов. Схема bLSAG также сокращает время, необходимое для подписания и верификации, за счёт удаления R и $\widetilde{K}$ из хеша, вычисляющего c_i.

Данный подход к связываемости будет более практичным применительно к Monero, чем алгоритм LSAG, так как он позволяет обнаруживать попытки двойной траты без введения каких-либо ограничений к количеству участников кольца.

3.3 Многоуровневые связываемые спонтанные анонимные групповые подписи (MLSAG)

Для того чтобы подписать транзакцию со множеством входов, понадобится m приватных ключей. В работах [26, 27] *Noether S.* и др. описывают многоуровневое обобщение схемы подписей bLSAG, которое может применяться в случае ряда $n \cdot m$ ключей, то есть, набора

$$R=\{K_{i,j}\} \text{ для } i \in \{1,2,\ldots,n\} \text{ и } j \in \{1,2,\ldots,m\}$$

для которого нам известны приватные ключи $\{k_{\pi,j}\}$, соответствующие поднабору $\{K_{\pi,j}\}$, для некоторого индекса $i=\pi$.[32]

32 Иначе схему MSAG можно представить как наличие m подколец с размером n, и в каждом кольце нам известен приватный ключ с индексом $i=\pi$ (всего $m \cdot n$ публичных ключей). Алгоритм подписи шифрует «стек» ключей на каждом этапе c, состоящий из одного ключа, взятого из каждого подкольца. Схема bLSAG представляет собой особый случай, если $m=1$.

Применение такого алгоритма решит нашу проблему наличия множества входов, но при этом подразумевается, что мы обобщим понятие связываемости.

Связываемость. Если какой-либо из приватных ключей $k_{\pi,j}$ используется в двух разных подписях, то эти подписи будут связаны автоматически.

Подпись

1. Вычислить образы ключей $\widetilde{K}=k_{\pi,j}H_p\left(K_{\pi,j}\right)$для всех $j\in\{1,2,\ldots,m\}$.
2. Сгенерировать случайные число $\alpha_j\in_R Z_l$ и $r_{i,j}\in_R Z_l$ для $i\in\{1,2,\ldots,n\}$ (за исключением $i=\pi$) и $j\in\{1,2,\ldots,m\}$.
3. Вычислить

$$c_{\pi+1}=H_n\left(m,\alpha_1 G,\alpha_1 H_p\left(K_{\pi,1}\right),\ldots,\alpha_m G,\alpha_m H_p\left(K_{\pi,m}\right)\right)$$

4. Для $i=\pi+1,\pi+2,\ldots,n,1,2,\ldots,\pi-1$, заменив $n+1\to 1$, вычислить

$$c_{i+1}=H_n\left(m,\left[r_{i,1}G+c_i K_{i,1}\right],\left[r_{i,1}H_p\left(K_{i,1}\right)+c_i\widetilde{K}_1\right],\ldots,\left[r_{i,m}G+c_i K_{i,m}\right],\left[r_{i,m}H_p\left(K_{i,m}\right)+c_i\widetilde{K}_m\right]\right)$$

5. Найти значение $r_{\pi,j}=\alpha_j-k_{\pi,j}c_\pi$ (mod l).

Подпись будет следующей: $\sigma(m)=\left(c_1,r_{1,1},\ldots,r_{1,m},\ldots,r_{n,1},\ldots,r_{n,m},\widetilde{K}_1,\ldots,\widetilde{K}_m\right)$.

Верификация

Верификация подписи производится следующим образом.

1. Для $i=1,\ldots,n$, заменив $n+1\to 1$, вычислить

$$c'_{i+1}=H_n\left(m,\left[r_{i,1}G+c_i K_{i,1}\right],\left[r_{i,1}H_p\left(K_{i,1}\right)+c_i\widetilde{K}_1\right],\ldots,\left[r_{i,m}G+c_i K_{i,m}\right],\left[r_{i,m}H_p\left(K_{i,m}\right)+c_i\widetilde{K}_m\right]\right)$$

2. Если $c'_1=c_1$, то подпись является действительной.

Почему это работает

Как и в случае с оригинальным алгоритмом LSAG, мы видим, что:
если $i\neq\pi$, то, очевидно, значения c'_{i+1} вычисляются, как описано в алгоритме подписи;
если $i=\pi$, то, так как $r_{\pi,j}=\alpha_j-k_{\pi,j}c_\pi$

$$\begin{aligned} r_{\pi,j}G &+c_\pi K_{\pi,j}=\left(\alpha_j-k_{\pi,j}c_\pi\right)G &+c_\pi K_{\pi,j}=\alpha_j G \\ r_{\pi,j}H_p\left(K_{\pi,j}\right)&+c_\pi\widetilde{K}_j=\left(\alpha_j-k_{\pi,j}c_\pi\right)H_p\left(K_{\pi,j}\right)+c_\pi\widetilde{K}_j=\alpha_j H_p\left(K_{\pi,j}\right)\end{aligned}$$

Другими словами, также будет действительным $c'_{\pi+1}=c_{\pi+1}$.

<u>Связываемость</u>

Если для создания какой-либо подписи повторно используется приватный ключ $k_{\pi,j}$, соответствующий образ изображения $\widetilde{K}_j$, передаваемый вместе с подписью, выявит его. Это соответствует нашему обобщённому определению связываемости.[33]

Требования к занимаемому пространству

Принимая во внимание сжатие точек, подпись MLSAG очевидно займёт всего

$$(1+nm+m)\cdot 32\,\textit{байта}$$

33 Как и в случае с LSAG, связанные подписи MLSAG не указывают, какой публичный ключ использовался для подписания. Темнее менее, если у каждого кольца есть только один общий ключ, то реально рабочий становится очевиден.

3.4 Кольцевые подписи Борромео

В последующих разделах этого отчёта наглядно продемонстрирована необходимость доказательства того, что суммы транзакции находятся в ожидаемом диапазоне. Это также можно реализовать при помощи кольцевых подписей. Однако, в данном конкретном случае необходимость в обеспечении связываемости подписей отсутствует, что позволяет нам выбрать более эффективный с точки зрения занимаемого места алгоритм.

В данном контексте и *исключительно с целью* доказательства диапазона суммы Monero использует[34] схему подписи, разработанную Г. Максвеллом (*G. Maxwell*) и описанную в работе [23]. Здесь в образовательных целях нами будет представлена полная версия схемы. В случае с Monero доказательства диапазона требуют использования подколец с 2 ключами, соответствующими каждому числу суммы, представленной двоичным методом. Это означает, что все $m_i=2$, поэтому схема Борромео может быть реализована в более простой форме.

Допустим, у нас есть ряд R публичных ключей $\left\{K_{i,j_i}\right\}$ для $i\in\left\{1,2,\ldots,n\right\}$ и $j_i\in\left\{1,2,\ldots,m_i\right\}$. Другими словами, $\left\{K_{i,j_i}\right\}$ подобен книжному стеллажу с публичными ключами с n полок, и на каждой i-ной полке находятся публичные ключи m_i в порядке от 1 до m_i. m_i может отличаться для каждого i, поэтому используется нижний индекс.

Кроме того, допустим, что для каждой i есть индекс π_i, позволяющиё подписанту узнать приватный ключ k_{i,π_i}, соответствующий K_{i,π_i}. Следуя аналогии, подписанту известен один приватный ключ для публичного ключа $j_i=\pi_i$ на каждой полке i книжного стеллажа.

Из чего следует, что m является хешем сообщения, которое должно быть подписано ключами $\left\{K_{i,j}\right\}$.

Подпись

1. Для каждой полки $i=1,\ldots,n$:
   (a) сгенерировать произвольное значение $\alpha_i\in_R Z_l$;
   (b) задать начальное число подкольца полки: задать $c_{i,\pi_i+1}=H_n\left(m,\alpha_i G,i,\pi_i\right)$;
   (c) построить первую половину подкольца, основанную на начальном числе: для $j_i=\pi_i+1,\ldots,m_i-1$ сгенерировать случайные числа $r_{i,j_i}\in_R Z_l$ и вычислить

$$c_{i,j_i+1}=H_n\left(m,\left[r_{i,j_i}G-c_{i,j_i}K_{i,j_i}\right],i,j_i\right)$$

2. Использовать последний публичный ключ на каждой полке для соединения всех подколец вместе: для $i=1,\ldots,n$ сгенерировать случайные числа $r_{i,m_i}\in_R Z_l$ и вычислить

$$c_1=H_n\left(\left[r_{1,m_1}G-c_{1,m_1}K_{1,m_1}\right],\ldots,\left[r_{n,m_n}G-c_{n,m_n}K_{n,m_n}\right]\right)$$

   Примечание: если какой-либо $\pi_i=m_i$, следует задать начальное число $\alpha_i G$ в соединяющем c_1.

3. Для каждой $i=1,\ldots,n$:
   (a) построить вторую половину подкольца на основе соединяющей части: для $j_i=2,\ldots,\pi_i-1$ сгенерировать случайные числа $r_{i,j_i}\in_R Z_l$ и вычислить [мы интерпретируем ссылки на $c_{i,1}$ как c_1]

$$c_{i,j_i+1}=H_n\left(m,\left[r_{i,j_i}G-c_{i,j_i}K_{i,j_i}\right],i,j_i\right)$$

34 В первой итерации «доказательств диапазона» Monero использовались агрегированные несвязываемые кольцевые подписи Шнорра (*Aggregate Schnorr Non-Linkable Ringм Signatures, ASNL*) [27]. По словам автора ASNL, она сокращается до некоторой кольцевой подписи Борромео [8], но так как последняя носит более общий характер и является более безопасной, именно она была выбрана в окончательном варианте реализации в ноябре 2016.

(b) связать концы подкольца вместе: задать r_{i,π_i} таким образом, чтобы $\alpha_i = r_{i,\pi_i} - c_{i,\pi_i} k_{i,\pi_i}$.

Подпись будет следующей:

$$\sigma = \left(c_1, r_{1,1}, \ldots, r_{1,m_1}, r_{2,1}, \ldots, r_{2,m_2}, \ldots, r_{n,m_n}\right)$$

Верификация

При известных m, R и σ верификация производится следующим образом.

1. Для $i = 1, \ldots, n$ и $j_i = 1, \ldots, m_i$ построить по кольцу:

$$L'_{i,j_i+1} = r_{i,j_i} G - c'_{i,j_i} K_{i,j_i}$$
$$c'_{i,j_i+1} = H_n\left(m, L'_{i,j_i+1}, i, j_i\right)$$

   Интерпретировать любую $c'_{i,1}$ как c_1.

2. Вычислить соединительную часть $c'_1 = H_n\left(L'_{1,m_1}, \ldots, L'_{n,m_n}\right)$.

Подпись является действительной, если $c'_1 = c_1$.

Почему это работает

1. Мы можем легко увидеть, что для $j \neq \pi_i$ и для всех i $c'_{i,j_i+1} = c_{i,j_i+1}$.
2. Если $j_i = \pi_i$ для всех i, то

$$L'_{i,\pi_i+1} = r_{i,\pi_i} G - c'_{i,\pi_i} K_{i,\pi_i}$$
$$¿ \left(\alpha_i + k_{i,\pi_i} c_{i,\pi_i}\right) G - c'_{i,\pi_i} K_{i,\pi_i}$$
$$¿ \alpha_i G + k_{i,\pi_i} c_{i,\pi_i} G - c'_{i,\pi_i} k_{i,\pi_i} G$$
$$¿ \alpha_i G$$

Другими словами, $c'_{i,\pi_i+1} + H_n\left(m, \alpha_i G, i, \pi_i\right) = c_{i,\pi_i+1}$.

Таким образом, мы можем прийти к выводу, что на этапе верификации идентифицируются действительные подписи.

4 Обязательства Педерсена

В общих чертах, криптографическая *схема обязательств* является способом предоставления доказательства суммы без раскрытия самой суммы.

Например, в игре с подбрасыванием монеты Элис может конфиденциально дать обязательства по одному результату (то есть, «назвать его»), до того, как Боб подбросит монету, опубликовав полученное значение, хешированное с секретными данными. После того, как он подбросит монету, Элис может объявить, какой результат она получила и доказать его, раскрыв секретные данные. После этого Боб может проверить её заявление.

Другими словами, предположим, что у Элис есть секретный ряд b, а значение, по которому она хочет дать обязательства — v. Она может просто хешировать $h=H(b,v)$ и сообщить Бобу h. Боб подбрасывает монету, Элис сообщает значение b Бобу и говорит ему, что она подтвердила v. После этого Боб вычисляет $h'=H(b,v)$. Если $h'=h$, то он знает, что Элис получила v до броска монеты.

4.1 Обязательства Педерсена

Обязательства Педерсена [29] обладают свойством *аддитивности*. Если $C(a)$ и $C(b)$ являются обозначением обязательств по сумме a и b, соответственно, то $C(a+b)=C(a)+C(b)$. Это свойство полезно при доказательстве сумм транзакций, так как любой может доказать, например, что входы равны выходам, не раскрывая имеющейся у него суммы.

К счастью, обязательства Педерсена легко реализовать, используя криптографию на основе эллиптических кривых, так как следующее известно заведомо:

$$aG+bG=(a+b)G$$

Очевидно, определяя обязательство просто как $C(a)=aG$, мы бы немедленно признали обязательства как нулевые (поскольку $0G=0$). Мы бы также смогли создать обманные таблицы обязательств, которые помогли бы нам распознать общие суммы a.

Для обеспечения информационно-теоретической[35] анонимности необходимо ввести секретный *скрывающий фактор* и ещё один генератор H, которые не позволят узнать, для какого значения γ будет действительным следующее равенство: $H=\gamma G$. Сложность решения дискретного логарифма гарантирует, что вычисление γ на основе H просто невозможно.

Следовательно, мы можем определить обязательство по сумме a как $C(x,a)=xG+aH$, где x является скрывающим фактором, который не даёт наблюдателям раскрыть a (например, если вы даёте обязательство $C(a=1)$, то его легко можно угадать и проверить).

Обязательство $C(x,a)$ является информационно-теоретически анонимным, так как существует множество возможных комбинаций x и a, которые будут иметь своим результатом C.[36] Если значение x действительно будет произвольным, у злоумышленника буквально не будет способа найти a [22].

В случае Monero $H=H_p(G)$.[37]

4.2 Обязательства Monero

35 Информационно-теоретическая безопасность означает такую защиту, что даже злоумышленник, обладающий бесконечно большой вычислительной мощностью, не сможет взломать шифр, поскольку не будет обладать достаточным количеством информации.

36 В основном, существует ТАКОЕ множество x' и a', что $x'+a'\gamma=x+a\gamma$. Лицу, публикующему обязательства, известна комбинация, но злоумышленник не может угадать, какая именно. Кроме того, даже лицо, публикующее обязательства, не сможет найти другой комбинации, не реши DLP для γ.

37 В кодовой базе Monero содержится функция *to_point()*, которая используется для присвоения скалярных величин точкам эллиптической кривой. В случае с доказательствами $H=¿_{point}(Keccak(G))$.

Владеть криптовалютой и владеть банковским счётом — это не одно и то же, так как во втором случае баланс клиента существует в виде единственного значения, указанного в базе данных. В первом случае человек, скорее, владеет группой *выходов* транзакций. У каждого выхода есть «сумма», а уже сумма всех выходов и считается балансом лица.

Для того чтобы отправить криптовалюту кому-то другому, создаётся транзакция. Транзакция использует старые выходы в качестве *входов* и направляет новые выходы получателям. Так как количество входов редко бывает равно заданному количеству выходов, большинство транзакций также включает в себя «сдачу», выход, который доставляет излишек обратно отправителю. Мы поговорим на эту тему в Главе 5.

В случае с Monero суммы транзакций скрыты при помощи технологии под названием RingCT, которая впервые была реализована в январе 2017 года. Несмотря на то, что верификаторы не знают, какое количество Moneroj (множественное число слова Monero, которое на языке эсперанто означает «деньги») содержится в каждом входе и выходе, им всё равно необходимо доказать, что сумма входов равна сумме выходов.

Другими словами, если у нас есть транзакция со входами, содержащими суммы $a_1,\dots,a_m$, и выходами с суммами $b_1,\dots,b_p$, то наблюдатель вполне обоснованно заподозрит, что:

$$\sum_j a_j - \sum_t b_t = 0$$

Так как обязательства являются аддитивными, сумма обязательств со входами и выходами также должна быть равна нулю[38]:

$$\sum_j C_{j,\in ¿} - \sum_t C_{t,out} = 0 ¿$$

Во избежание возможной идентификации отправителя, *Shen Noether* предлагает [26] верифицировать равенство суммы обязательства определённому ненулевому значению:

$$\sum_j C_{j,\in ¿} - \sum_t C_{t,out} = zG ¿$$
$$\sum_j \left(x_j G + a_j H\right) - \sum_t \left(y_t G + b_t H\right) = zG$$
$$\sum_j x_j - \sum_t y_t = z$$

Причины, которые делают весь этот процесс полезным, становятся ясны в Главе 5, в которой рассматривается структура транзакций.

4.3 Доказательства диапазона

Одной из проблем аддитивных обязательств является то, что если у нас есть обязательства $C(a_1)$, $C(a_2)$, $C(b_1)$ и $C(b_2)$, и мы намерены использовать их с целью доказательства, что $(a_1+a_2)-(b_1+b_2)=0$, то эти обязательства будут применяться, только если одно из значений в уравнении будет отрицательным.

Например, у нас может быть $a_1=6$, $a_2=5$, $b_1=21$ и $b_2=-10$.

$$(6+5)-(21\pm 10)=0$$

где

$$21G \pm 10G = 21G + (l-10)G = (l+11)G = 11G$$

38 Согласно изложенному в Разделе 2.2.1, мы можем выделить точку, инвертировав её координаты. Если $P=(x,y), -P=(x,-y)$. Также следует помнить о том, что отрицательные значения элементов поля вычисляются (mod q), поэтому ($-y$ (mod q)).

Мы могли бы сохранить 21 выход и отбросить -10 выход, эффективно создав таким на 10 Monero больше, чем вложили.

Решением данной проблемы в случае Monero является доказательство каждой суммы выхода в определённом диапазоне при помощи схемы подписей Борромео, описанной в разделе 3.4.

При наличии обязательства $C(b)$ со скрывающим фактором y_b для суммы b, используем двоичное представление $(b_0, b_1, \ldots, b_{k-1})$, чтобы получить

$$b = b_0 2^0 + b_1 2^1 + \ldots + b_{k-1} 2^{k-1}$$

Сгенерируем случайные числа $y_0, \ldots, y_{k-1} \in_R Z_l$, которые будем использовать в качестве скрывающих факторов. Также определим обязательства Педерсона для каждого b_i, $C_i = y_i G + b_i 2^i H$, и выведем приватные ключи $\{C_i, C_i - 2^i H\}$.

Очевидно, что один из этих публичных ключей будет равен $y_i G$:

если $b_i = 0$, то $C_i \quad ¿\, y_i G + 0H \quad = y_i G$

если $b_i = 1$, то $C_i - 2^i H \quad ¿\, y_i G + 2^i H - 2^i H \; = y_i G$

Другими словами, скрывающий фактор y_i всегда будет приватным ключом, соответствующим одной из точек $\{C_i, C_i - 2^i H\}$. Одна из этих точек является *обязательством нуля*, поскольку либо $b_i 2^i = 0$, либо $b_i 2^i - 2^i = 0$. Мы может доказать, что сумма b в выходе транзакции находится в диапазоне $[0, \ldots, 2^k - 1]$, подписав её, используя схему кольцевых подписей Борромео, описанную в Разделе 3.4, с кольцом публичных ключей:

$$\{\{C_0, C_0 - 2^0 H\}, \ldots, \{C_{k-1}, C_{k-1} - 2^{k-1} H\}\}$$

где нам известны приватные ключи $\{y_0, \ldots, y_{k-1}\}$, соответствующие каждой паре.

Полученная подпись будет следующей: $\sigma = (c_1, r_{0,1}, r_{0,2}, r_{1,1}, \ldots, r_{k-1,2})$.

4.4 Доказательства диапазона в блокчейне

В контексте Monero мы используем доказательства диапазона для доказательства действительности суммы в выходах каждой транзакции.

Верификаторам транзакций придётся проверить равенство суммы обязательства по доказательству диапазона каждого выхода C_i обязательству по самой сумме C_b. Для того, чтобы это сработало, нам необходимо модифицировать наше определение скрывающих факторов y_i: ряд $y_0, \ldots, y_{k-2} \in_R Z_l$, определить $y_{k-1} = y_b - \sum_{i=0}^{k-2} y_i$. Теперь у нас есть следующее уравнение:

$$\sum_{i=0}^{k-1} C_i = C_b$$

В блокчейне мы сохраним только обязательства по доказательству диапазона / ключи C_i, обязательство выхода C_b, а также члены подписи σ. Сообщество майнеров с лёгкостью сможет вычислить $C_i - 2^i H$ и верифицировать кольцевую подпись Борромео для каждого выхода.

Получателю или любой другой стороне совсем необязательно знать скрывающие факторы y_i, так как они служат исключительно для доказательства того, что сумма нового выхода находится в допустимом диапазоне.

Так как для создания подписи схема кольцевых подписей Борромео требует знания y_i, любая третья сторона, верифицирующая её, может убедиться в том, что каждое подкольцо содержит обязательство нуля, так как итоговые суммы должны укладываться в диапазон, и деньги не будут создаваться искусственным образом.

5 Транзакции Monero

5.1 Ключи пользователей

В отличие от Bitcoin Monero использует два набора приватных/публичных ключей, k^v, K^v и k^s, K^s, которые генерируются, как описано в Разделе 2.2.2.[39]

Адрес пользователя представлен парой публичных ключей $\left(K^v, K^s\right)$. Приватными ключами будет соответствующая пара $\left(k^v, k^s\right)$.

Использование двух наборов ключей делает возможным разделение функций. Обоснованность такого подхода станет очевидной позже в этой главе, а сейчас предлагаю называть приватный ключ k^v *ключом просмотра*, а ключ k^s *ключом траты*. Пользователь сможет использовать свой ключ просмотра для того, чтобы определить, адресован выход ему или кому-либо другому, а ключ траты позволит ему отправлять этот выход в транзакцию.

5.2 Одноразовые (скрытые) адреса

У каждого пользователя Monero есть свой публичный адрес, которым он может поделиться с другими пользователями, чтобы они использовали его в выходах транзакции. Этот адрес никогда не используется напрямую. Вместо этого происходит обмен, как в случае с использованием алгоритма Диффи-Хеллмана. В результате воздаётся уникальный *скрытый адрес* для каждого выхода транзакции, который должен быть выплачен пользователю. Таким образом, даже те внешние наблюдатели, которым известны все публичные адреса пользователей, не смогут идентифицировать пользователя, получившего какой-либо из выходов определённой транзакции.

Давайте рассмотрим эту концепцию более подробно на примере очень простой транзакции с одним входом и одним выходом — проведении платежа от Элис к Бобу.

У Боба есть приватные/публичные ключи $\left(k_B^v, k_B^s\right)$ и $\left(K_B^v, K_B^s\right)$, и Элис известны его публичные ключи. Транзакция могла бы происходить следующим образом (см. [34]):

1. Элис генерирует случайное число r таким образом, чтобы $1<r<l$, и вычисляет одноразовый публичных ключ $K^o = H_n\left(r\,K_B^v\right)G + K_B^s$.
2. Элис устанавливает K^o в качестве адресата платежа, добавляет значение rG к данным транзакции и передаёт их в сеть.
3. Боб получает данные и видит значения rG и K^o. Он может вычислить $k_B^v rG = r\,K_B^v$. Затем он также может вычислить $K_B^{'s} = K^o - H_n\left(r\,K_B^v\right)G$. Как только он увидит, что $K_B^{'s} = K_B^s$, он будет знать, что транзакция адресована ему.

   Приватный ключ k_B^v называется *ключом просмотра*, поскольку любой, у кого он есть (вместе с публичным ключом траты K_B^s Боба), может вычислить $K^{'o}$ для каждого выхода транзакции в сети, у «просмотреть», какие из выходов адресованы Бобу.
4. Одноразовыми ключами для выхода будут

$$K^o = H_n\left(r\,K_B^v\right)G + k_B^s G = \left(H_n\left(r\,K_B^v\right) + k_B^s\right)G$$

---

39 В настоящее время чаще всего ключ просмотра k^v равен $H_n\left(k^s\right)$. Это означает, что пользователю просто нужно сохранить свой ключ траты k^s, чтобы получить доступ ко всем выходам, которые у него имеются. Ключ траты обычно представлен как ряд из 25 слов (где 25-е слово является контрольной суммой). Другими, менее популярными методами являются: генерирование k^v и k^s как отдельных случайных чисел, или же генерирование случайной мнемонической фразы a, состоящей из 12 слов, где $k^s = sc_{reduce}32\left(Keccak\left(a\right)\right)$ и $k^v = sc_{reduce}32\left(Keccak\left(Keccak\left(a\right)\right)\right)$.

$$k^o = H_n\left(r K_B^v\right) + k_B^s$$

Несмотря на то, что Элис может вычислить публичный ключ адреса K^o, она не может вычислить соответствующий приватный ключ k^o, так как для этого ей необходимо знать либо ключ траты k_B^s Боба, либо решить дискретный логарифм для $K_B^s = k_B^s G$, что является довольно сложной задачей. Как станет ясно далее в этой главе, не зная k^o, Элис не сможет вычислить образ ключа выхода и не будет знать наверняка, потратил ли Боб выход, который она отправила ему.

Третья сторона, обладающая *ключом просмотра* Боба, может проверить, что выход адресован Бобу. При этом, не зная *ключа траты*, третья сторона не сможет ни потратить этот выход, ни узнать, когда он был потрачен. Также третья сторона не сможет использовать приватный ключ k^o одноразового адреса для подписи, а также не сможет создать образ ключа адреса.

Такая третья сторона может являться доверенным хранителем, аудитором, налоговым органом и т. д. Кем угодно, кто обладает доступом к истории транзакций пользователя, но кто не обладает какими-либо большими правами. Эта третья сторона также сможет расшифровать суммы, как это описано в Разделе 5.6.1.

5.2.1 Транзакции со множеством выходов

Большинство транзакций содержит более одного выхода. Если что-либо другое отсутствует, то «сдача» пересылается обратно отправителю.

Отправители Monero генерируют только одно случайное значение r. Значение rG обычно называют *публичным ключом транзакции*, и он публикуется в блокчейне.

Чтобы гарантировать, что все адреса выходов в транзакции при наличии p выходов являются разными, даже в случаях, когда один и тот же адрес применяется дважды, Monero использует индексы выходов. Каждый выход транзакции имеет индекс $t \in 1, \ldots, p$. Приложив это значение к совместно используемым секретным данным перед хешированием, можно гарантировать, что полученные скрытые адреса будут уникальными:

$$K_t^o = H_n\left(r K_t^v\right) G + K_t^s = \left(H_n\left(r K_t^v, t\right) + k_t^s\right) G$$
$$k_t^o = H_n\left(r K_t^v, t\right) + k_t^s$$

5.3 Подадреса

Пользователи Monero могут генерировать подадреса на основе каждого адреса [25]. Средства, отправляемые на подадреса, могут быть просмотрены и потрачены при помощи ключей просмотра и траты основного адреса. Аналогичным образом: под одной учётной записью пользователя в онлайн банке может быть находиться балансов, соответствующих кредитным картам и вкладам, и ко всем ним можно получить доступ через одну и ту же точку — учётную запись владельца.

Подадреса удобны с точки зрения получения средств в одно и то же место, если пользователь не хочет связывать все свои действия вместе, публикуя/используя один и тот же адрес. Как можно будет увидеть, наблюдателю придётся решить DLP, чтобы определить, был ли выведен конкретный подадрес на основе определённого адреса [25].

Подадреса также полезны с точки зрения дифференциации полученных выходов. Например, если Элис захочет купить яблоко у Боба во вторник, то Боб может выписать квитанцию об оплате, в которой будет указан покупаемый предмет, а также указать подадрес этой квитанции. Затем Боб попросит Элис использовать этот подадрес, когда она будет отправлять ему деньги. Таким образом, Боб может связать деньги, которые он получает, с яблоком, которое он продал. Мы исследуем ещё один способ того, как отличить полученные выходы, в следующем разделе.

Боб генерирует свой подадрес i ($i = 1, 2, \ldots$) на основе своего адреса как пару публичных ключей $\left(K^{v,i}, K^{s,i}\right)$:

$$K^{s,i} = K^s + H_n\left(k^v, i\right)G$$
$$K^{v,i} = k^v K^{s,i}$$

Таким образом,

$$K^{v,i} = k^v\left(k^s + H_n\left(k^v, i\right)\right)G$$
$$K^{s,i} = ¿\left(k^s + H_n\left(k^v, i\right)\right)G$$

5.3.1 Отправка средств на подадрес

Допустим, Элис собирается отправить Бобу деньги на его подадрес $\left(K_B^{v,1}, K_B^{s,1}\right)$ путём проведения простой транзакции с одним входом и одним выходом.

1. Элис генерирует случайное число r таким образом, чтобы $1<r<l$, и вычисляет одноразовый публичных ключ $K^o = H_n\left(r\,K_B^{v,1}\right)G + K_B^{s,1}$.
2. Элис устанавливает K^o в качестве адресата платежа, **добавляет значение** $r\,K_B^{s,1}$ **к данным транзакции** и передаёт их в сеть.
3. Боб получает данные и видит значения $r\,K_B^{s,1}$ и K^o. **Он может вычислить** $k_B^v r\,K_B^{s,1} = r\,K_B^{v,1}$. Затем он также может вычислить $K_B^{'s,1} = K^o - H_n\left(r\,K_B^{v,1}\right)G$. Как только он увидит, что $K_B^{'s,1} = K_B^{s,1}$, он будет знать, что транзакция адресована ему.

   Для того чтобы найти выходы транзакции, предназначенные для его подадресов, Бобу нужен только его приватный ключ просмотра k_B^v.
4. Одноразовыми ключами для выхода будут

$$K^o = H_n\left(r\,K_B^{v,1}\right)G + K_B^{s,1}G = \left(H_n\left(r\,K_B^{v,1}\right) + k_B^{s,1}\right)G$$
$$k^o = H_n\left(r\,K_B^{v,1}\right) + k_B^{s,1}$$

Теперь публичный ключ транзакции Элис предназначен для Боба ($rK_B^{s,1}$ вместо rG). Если Элис создаст транзакцию с m выходов, где, по крайней мере, один выход будет предназначен для подадреса, ей потребуется создать уникальный публичный ключ транзакции для каждого выхода $t \in 1,\ldots,m$. Другими словами, если Элис отправляет средства на подадрес Боба $\left(K_B^{v,1}, K_B^{s,1}\right)$ и на адрес Кэрол $\left(K_C^v, K_C^s\right)$, ей придётся включить в данные транзакции два публичных ключа транзакции $\left\{r_1 K_B^{s,1}, r_2 G\right\}$.[40]

5.4 Интегрированные адреса

Для того чтобы отличить между собой полученные выходы, получатель может попросить у отправителей включить в данные транзакции *идентификатор (ID) платежа*. Например, если Элис захочет купить яблоко у Боба во вторник, то Боб может выписать квитанцию об оплате, в которой будет указан покупаемый предмет, а также попросить Элис включить ID платежа, указанный в квитанции, когда она будет отправлять ему деньги. Таким образом, Боб может связать деньги, которые он получает, с яблоком, которое он продал.

Отправители могут сообщать ID платежа простым текстом, но ручной ввод ID в данные транзакции является неудобным способом, а также представляет собой определённую опасность для анонимности получателей, которые могут случайно раскрыть свои действия. В случае с Monero получатели могут интегрировать ID платежа в свои адреса и передавать такие *интегрированные адреса*, содержащие (K^v, K^s, ID платежа), отправителям. ID платежа могут быть технически интегрированы в любой вид адреса, включая обычные адреса, подадреса и адреса с мультиподписями.[41]

40 В случае с Monero подадреса имеют префикс «8», который отделяет их от адресов, которые имеют префикс «4». Это помогает отправителям выбрать правильную процедуру при построении транзакций.

Отправители, которые направляют выходы на интегрированные адреса, могут зашифровать ID платежа, используя общие секретные данные K_t^v, индекс выхода t или операцию XOR, чтобы получатель мог потом смог расшифровать идентификатор при помощи соответствующего публичного ключа транзакции или другой процедуры XOR [3]. Шифрование ID платежа подобным образом позволяет отправителям доказать, что они провели определённую транзакцию (то есть, в случае аудита, возмещения средств и других подобных случаях).

Двоичный оператор XOR

Двоичный оператор XOR оценивает два аргумента и возвращает истинный аргумент, если один из них, но не оба, таковым является [7]. Вот таблица истинности:

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F

В контексте компьютерной науки операция XOR эквивалента поразрядовому добавлению по модулю 2. Например, XOR двух пар:

$$XOR(\{1,1\},\{1,0\}) = \{1+1,1+0\} \pmod 2$$
$$¿\{0,1\}$$

В случае с предыдущим примером в каждом случае получается один и тот же результат: $XOR(\{1,1\},\{1,0\})$, $XOR(\{0,0\},\{0,1\})$, $XOR(\{1,0\},\{1,1\})$ или $XOR(\{0,1\},\{0,0\})$. Для одного и того же выхода существуют 2-битные комбинации входов XOR, поэтому при использовании входа $A \in_R (1,\ldots,2^{bits})$ наблюдатель, которому известно $C = XOR(A,B)$, не может получить какой-либо информации о B.

В то же самое время, любой, кому известны два элемента A,B,C, где $C = XOR(A,B)$, может вычислить третий элемент: $A = XOR(B,C)$. XOR показывает, являются два элемента разными или одинаковыми, поэтому достаточно знать C и B, чтобы раскрыть A. Тщательное изучение таблицы истинности подтверждает это.

Шифрование

Отправитель зашифровывает каждый ID платежа[42] для его включения в данные транзакции

$$k_{stealth} = H_n(r K_t^v, t)$$
$$k_{payment\ ID} = k_{stealth} \rightarrow сокращение\ до\ длины\ ID\ платежа\ в\ битах$$
$$зашифрованный\ ID\ платежа = XOR(k_{payment\ ID}, ID\ платежа)$$

Выход хеш-функции H равномерно распределяется по всему ряду возможных выходов. Другими словами, для некоторого входа A, $H(A) \in_R^D S_H$, где S_H является рядом возможных выходов H. $\in_R^D$ используется нами для обозначения того, что функция является

41 ID платежа и интегрированные адреса, использованные в версии v0.12 популярного GUI Monero, были раскритикованы и будут иметь слабую поддержку со стороны сообщества разработчика Monero в будущем. так как наблюдатель может увидеть разницу между транзакциями при использовании и без ID платежа какого-либо вида, их использование делает историю транзакций Monero менее единообразной. Так как в тех же целях есть возможность использования подадресов (более известных в качестве «одноразовых адресов»), ID платежа могут оказаться излишеством. Также следует отметить, что интегрированные адреса пока использовались только с обычными адресами.

42 В случае Monero ID платежа в интегрированных адресах традиционно имеют длину 64 бита, в то время как независимые ID платежа, написанные простым текстом, имеют длину 256 бит.

детерминировано случайной — $H(A)$ всегда имеет один и тот же результат, но её выход эквивалентен случайному числу.[43]

Расшифровка

Получатель *t* может найти его *t* идентификатор платежа, используя свой ключ просмотра и публичный ключ транзакции *rG* следующим образом:

$$k_{stealth} = H_n\left(k_t^v rG, t\right)$$

$$k_{payment\ ID} = k_{stealth} \rightarrow сокращение\ до\ длины\ ID\ платежа\ в\ битах$$

$$ID\ платежа = XOR\left(k_{payment\ ID}, зашифрованный\ ID\ платежа\right)$$

Подобным образом, отправители могу расшифровывать ID платежа, который они ранее зашифровали, путём пересчёта общих секретных данных $k_{stealth} = H_n\left(rK_t^v, t\right)$.

5.5 Типы транзакций

Monero является криптовалютой, постоянно находящейся в разработке. Структуры транзакций, протоколы и криптографические схемы развиваются по мере появления новых задач или угроз.

В данном отчёте мы уделяем основное внимание протоколу кольцевых конфиденциальных транзакций (*Ring Confidential Transactions*), также известному как RingCT, так как он используется в текущей версии Monero. Протокол RingCT обязателен для использования всеми новыми транзакциями Monero, поэтому мы не станем описывать унаследованные типы транзакций, хотя они до сих пор частично поддерживаются.

В данной главе мы опишем следующие типы транзакций: RCTTypeFull и RCTTypeSimple. Первая категория (см. Раздел 5.6) тесно связана с идеями, изложенными *S. Noether* и др. в работе [27]. К моменту завершения этого отчёта авторы, вероятнее всего, уже заменят оригинальную схему транзакций CryptoNote.

Тем не менее, в случае с транзакциями со множеством выходов в результате использования схемы подписей, сформулированной в той работе, мог возникнуть риск отслеживаемости. Это станет понятно, когда мы дойдём до технических подробностей, но если описать ситуацию кратко: если какой-либо из потраченных выходов можно будет идентифицировать, остальные потраченные выходы также будут идентифицированы. Это повлияет на отслеживаемость денежных потоков, и не только на источник транзакции, но и на блокчейн в целом.

Во избежание такого риска лабораторией Monero Research Lab было принято решение использовать связанную, но всё же другую схему подписей для транзакций со множеством подписей. В таких случаях используется тип транзакции RCTTypeSimple (см. Раздел 5.7). Его основное отличие заключается в том, что, как мы увидим далее, каждый вход подписывается независимо.

В Разделе 5.8 нами приводится краткое описание концепции данных транзакций.

---

43 Хеш-функции также известны как случайные посредники (*random oracles*) [23]. «В криптографии… посредником может являться любая система, которая даёт некоторую дополнительную информацию по системе, которая в противном случае была бы недоступной» [1]. Например, доказательство безопасности в случае защиты схемы подписей от подделки, описанное в работе [21], подразумевает доступ противной стороны к посреднику, который может выдать действительные подписи для любого заданного сообщения, а также к использованию любого заданного публичного ключа (модель адаптивно выбираемого сообщения и адаптивно выбираемого публичного ключа). В том случае, если противная сторона не сможет подделать подпись после обработки её запросов посредником, значит схему действительно невозможно подделать. Система называется посредником (*a oracle также переводится как «оракул»*), так как её использование подобно обращению к Богу за помощью.

Хеш-функции являются случайными посредниками, так как они работают также, как если бы вы просили кого-то назвать случайное число. Случайные посредники полезны с точки зрения криптографии, поскольку они могут быть публично проверены.

5.6 Кольцевые конфиденциальные транзакции типа RCTTypeFull

По умолчанию в текущей кодовой базе в отношении транзакций с одним входом используется именно этот тип схемы подписей. Сама схема допускает применение к транзакциям со множеством входов, но когда она появилась, лабораторией Monero Research Lab было принято решение, что лучше использовать её только с транзакциями с одним входом. В случае с транзакциями со множеством существующие на данный момент кошельки Monero используют схему RCTTypeSimple, которая будет описана далее.

Как нам кажется, решение ограничить применение схемы RCTTypeFull транзакциями с одним входом было принято поспешно, и оно может измениться в будущем, если алгоритм выбора дополнительных смешиваемых выходов будет улучшен, а размер кольца увеличен. Помимо этого, оригинальное описание, которое приводит *S. Noether* в работе [27], не предусматривает ограничений подобного рода. В любом случае, это не строгое ограничение. Альтернативные кошельки могут использовать эту схему для подписания транзакций, независимо от количества входов, которое в них содержится.

Вот поэтому мы и решили описать эту схему, как если бы она предназначалась и для транзакций со множеством входов.

Реальный пример RCTTypeFull транзакции со всеми её компонентами приводится в Приложении А.

5.6.1 Обязательства по сумме

Вернёмся к Разделу 4.2 и вспомним, что мы определили обязательство для суммы b выхода как:

$$C(b) = yG + bH$$

В контексте Monero получатели выходов должны иметь возможность просмотреть соответствующие суммы. Это означает, что получателю должен быть сообщён скрывающий фактор yG.

Monero использует общие секретные данные rK_B^v Диффи-Хеллмана. В случае с каждой отдельно взятой транзакцией в блокчейне каждый её выход $t \in 1, \ldots, p$ имеет два связанных с ним значения, которые называются *маской* (*mask*) и *суммой* (*amount*), которые соответствуют[44]

$$mask_t = y_t + H_n(r K_B^v, t)$$
$$amount_t = b_t + H_n(H_n(r K_B^v, t))$$

Получатель, Боб, сможет вычислить скрывающий фактор y_t и сумму b_t, используя *публичный ключ транзакции* $r G$ и свой ключ просмотра k_B^v. Он также может проверить соответствие обязательства $C(y_t, b_t)$ в данных транзакции, которое далее мы обозначим как C_t^b, сумме, которая у него имеется.

В более общем смысле, любая третья сторона, у которой есть доступ к *ключу просмотра* Боба, сожжет расшифровать суммы в его выходах и убедиться в их соответствии связанным с ними обязательствам.

5.6.2 Нулевые обязательства

44 Как и в случае со скрытым адресом K^o индекс выхода t добавляется к каждому хешу в каждой паре «маска/сумма». Это гарантирует, что выходы, направленные по одному адресу, будут в безопасности. Кроме того, «сумма» содержит дополнительный хеш H_n, который не позволяет произвести статистический анализ данных блокчейна, который может способствовать отслеживанию потоков валюты и в целом разделить значения «маски/суммы».

Допустим, отправителем транзакции ранее были получены суммы $a_1,\dots,a_m$ из различных выходов, направленных на одноразовый адрес $K^o_{\pi,1},\dots,K^o_{\pi,m}$ с обязательствами по сумме $C^a_{\pi,1},\dots,C^a_{\pi,m}$.

Этому отправителю известны приватные ключи $k^o_{\pi,1},\dots,k^o_{\pi,m}$, соответствующие одноразовым адресам (см. Раздел 5.2). отправителю также известны скрывающие факторы x_j, используемые обязательствами $C^a_{\pi,j}$ (см. Раздел 5.6.1).

Транзакция состоит из входов $a_1,\dots,a_m$ и выходов $b_1,\dots,b_p$, в результате чего получаем $\sum_{j=1}^{m} a_j - \sum_{t=1}^{p} b_t = 0$.

Отправитель повторно использует обязательства из предыдущих выходов, $C^a_{\pi,1},\dots,C^a_{\pi,m}$, и создают обязательства для $b_1,\dots,b_p$. Обозначим эти обязательства как $C^b_1,\dots,C^b_p$.

Как уже говорилось в Разделе 4.2, сумма обязательств не будет совершенно равна 0, но будет соответствовать точке кривой zG:

$$\sum_j C^a_{\pi,j} - \sum_t C^b_{\pi,t} = zG$$

Отправителю будет известно значение z, что позволит ему создать подпись по этому *нулевому обязательству*.

Фактически, z выводится из скрывающих факторов если и только в том случае, когда количество входов равно количеству выходов (как было описано в Разделе 4.1., мы не знаем γ в $H = \gamma G$):

$$\sum_{j=1}^{m} C^a_{\pi,j} - \sum_{t=1}^{p} C^b_{\pi,t}$$

$$¿ \sum_j x_j G - \sum_t y_t G + \left(\sum_j a_j - \sum_t b_t\right) H$$

$$¿ \sum_j x_j G - \sum_t y_t G$$

$$¿ zG$$

5.6.3 Подпись

Отправитель выбирает из блокчейна v наборов размера m дополнительных несвязанных адресов и их обязательств, соответствующих очевидно неотправленным выходам.[45] Он смешивает адреса в *кольце*, используя собственные адреса непотраченных выходов m, добавляя ложные нулевые обязательства, следующим образом:

45 В случае с Monero стандартный выбор набора «дополнительных несвязанных адресов» происходит по следующему принципу: 50% берётся из тех, что были использованы за 1,8 дня до даты траты транзакции по умолчанию, что обычно составляет 10 блоков, а 50% берутся из остальной части блокчейна. Каждый сегмент имеет треугольную вероятность схождения в отметке 1,8 дня, в которой существует двойная вероятность того, что выход, реализованный 1,8 дня назад, будет выбран как выход, использованный 0,9 дня назад. Чтобы потратить выход типа X находятся все остальные выходы типа X (то есть, выходы RingCT) и на основе такого распределения выбираются члены кольца. Треугольная вероятность обеспечивается путём прокрутки случайного количества всех членов ложного кольца с их нормированием , извлечением квадратного корня, умножения на количество удовлетворяющих критерии выходов типа X и использования выхода в этом индексе группы (если стороны треугольника перевёрнуты, необходимо перевернуть и индекс).

$$R=\left\{\left\{K^o_{1,1},\ldots,K^o_{1,m}\right\},\left(\sum_j C_{1,j}-\sum_t C^b_t\right)\right\}$$
$$\ldots$$
$$\left\{K^o_{\pi,1},\ldots,K^o_{\pi,m},\left(\sum_j C^a_{\pi,j}-\sum_t C^b_t\right)\right\}$$
$$\ldots$$
$$\left\{K^o_{v+1,1},\ldots,K^o_{v+1,m},\left(\sum_j C_{v+1,j}-\sum_t C^b_t\right)\right\}$$

Взглянув на структуру кольца ключа мы увидим, что если

$$\sum_j C^a_{\pi,j}-\sum_t C^b_t=0$$

значит, каждый наблюдатель распознает ряд адресов $\left\{K^o_{\pi,1},\ldots,K^o_{\pi,m}\right\}$ как ряд, используемый в качестве входов, и следовательно, поток валюты будет отслеживаемым.

Учитывая всё это, мы можем сделать выводы о практичности использования zG. Все члены обязательства в R выдают какую-либо точку на эллиптической кривой, и для π таким членом является zG. Это позволяет нам создать подпись MLSAG (см. Раздел 3.3) для R.

Подпись MLSAG для входов. Приватными ключами для $\left\{K^o_{\pi,1},\ldots,K^o_{\pi,m},\left(\sum_j C^a_{\pi,j}-\sum_t C^b_t\right)\right\}$ являются $K^o_{\pi,1},\ldots,K^o_{\pi,m}$, z, которые известны отправителю. При таком сценарии MLSAG не использует образ ключа для нулевого обязательства zG. Это означает, что вычисление и верификация подписи исключает член $r_{i,m+1}H_p\left(K_{i,m+1}\right)+c_i\widetilde{K}_z$.

Сообщение m, подписанное в MLSAG входа, в значительной степени является хешем всей информации транзакции, *за исключением* самой подписи MLSAG.[46] Это гарантирует, что транзакции будут защищены от вмешательства с точки зрения как авторов транзакции, так и верификаторов.

Доказательства диапазона выходов. Во избежание неясности суммы в выходов, описанной в Разделе 4.3, отправитель должен также использовать схему подписей Борромео (см. Раздел 3.4), чтобы подписать диапазон суммы для каждого выхода $t\in 1,\ldots,p$. Ни одно из сообщений m не подписывается подписью Борромео.

Доказательства диапазона для сумм во входах не требуются, поскольку они либо явно обозначены (как в случае с комиссиями за транзакции и наградами за блок), либо их диапазон был доказан при изначальном создании выходов.

В текущей версии программного обеспечения Monero каждая сумма выражена как 64-битный номер фиксированной точки. Это означает, что данные каждого доказательства диапазона должны содержать 64 обязательства и 2 · 64 + 1 членов подписи.

5.6.4 Комиссии за проведение транзакций

Обычно выходы транзакций в целом *ниже*, чем входы транзакций, что обеспечивает возможность выплаты комиссий майнерам. Суммы комиссий за проведение транзакций хранятся в форме обычного текста в данных транзакции, передаваемых по сети. Майнерами для себя может быть создан дополнительный выход с комиссией. Эта сумма комиссии также должна быть преобразована в обязательство, чтобы верификаторы смогли подтвердить сумму транзакции как нулевую.

---

46 Фактическим сообщением является
$m=H\left(H\left(tx_{prefix}\right),H\left(ss\right),H\left(подписи\,доказательства\,диапазона\right)\right)$, где:
$tx_{prefix}=\left\{версия\,транзакции\left(то\,есть,ringCT=2\right)\right\}$, входы {офсеты ключей, образ ключа}, выходы {одноразовые адреса}, дополнительные данные {публичный ключ транзакции, ID платежей, зашифрованные ID платежей, прочее}};
ss = {тип подписи (простая пили полная), комиссия за транзакцию, псевдо обязательства выходов для входов, ecdhInfo (маски и суммы), обязательства выходов}.
См. Приложение A и B, в которых разъясняется данная терминология.

Решением является вычисление обязательства по комиссии f без маскировки при помощи какого-либо скрывающего фактора. То есть, $C(f)=fH$, где f указывается обычным текстом.

Сеть проверяет подпись MLSAG в R следующим образом:

$$\left(\sum_j C_{i,j}-\sum_t C_t^b\right)-f\,H$$

Это работает, поскольку является нулевым обязательством:

$$\left(\sum_j C_{\pi,j}-\sum_t C_t^b\right)-f\,H=zG$$

5.6.5 Исключение возможности двойной траты

Подпись MLSAG (см. Раздел 3.3) содержит образы $\widetilde{K}_j$ приватных ключей $k_{\pi,j}$. Важным свойством каждой криптографической схемы подписей является невозможность её подделки с высокой степенью вероятности. Следовательно, во всех практических смыслах, мы можем допустить, что образы ключей подписи должны детерминировано выводиться из действительных приватных ключей.

Сети необходимо только проверить, чтобы образы ключей, включённые в подписи MLSAG (соответствующие входам и вычисленные как $\widetilde{K}_j^o=k_{\pi,j}^o\,H_p\left(K_{\pi,j}^o\right)$), не встречались ранее в других транзакциях.[47] Если встречались, то вы можете быть уверенными в том, что столкнулись с попыткой повторной траты выхода $C_{\pi,j}^a$, адресованного $K_{\pi,j}^o$.

Если кто-нибудь попытается потратить $C_{\pi,j}^a$ дважды, то этому человеку понадобится открыть индекс π для обеих транзакций, где он появляется. А это имеет два последствия: 1) все выходы с индексом π в первой транзакции откроются как его реальные входы, и 2) все выходы с индексом π во второй транзакции откроются как ранее не потраченные. Второе последствие является проблемой, даже при том, что все майнеры отклонят транзакцию с попыткой двойной траты.

Эти последствия могут ослабить сеть, лишив её тех преимуществ, которые дают кольцевые подписи, а также именно они являются причиной того, что схема RCTTypeFull используется только в случае с транзакциями с одним входом. Другая главная причина состоит в том, что криптоаналитикам будет известно, что, в целом, все реальные входы совместно используют индекс.

5.6.6 Требования к занимаемому месту

Подпись MLSAG (входы)

Из Раздела 3.3 мы помним, что подпись MLSAG в данном контексте будет выражена как

$$\sigma(m)=\left(c_1,r_{1,1},\ldots,r_{1,m+1},\ldots,r_{v+1,1},\ldots,r_{v+1,m+1},\widetilde{K}_1^o,\ldots,\widetilde{K}_m^o\right)$$

В силу наследия, оставленного применением CryptoNote, значения $\widetilde{K}_j^o$ не считаются частью подписи, а скорее являются *образами* приватных ключей $k_{\pi,j}^o$. Эти *образы ключей* обычно хранятся отдельно в рамках структуры транзакции, так как они используются для выявления атак с попыткой двойной траты.

---

47 Верификаторы также должны проверить, является ли образ изображения членом подгруппы генератора (см. Раздел 2.2.1), убедившись, что $l\widetilde{K}_j^o=0$, так как иногда в образу изображения можно добавить точку EC с порядком подгруппы, равным множеству l и при этом создать верифицируемую подпись. Более подробную информацию можно найти в работе [30].

Учитывая всё это, а также сжатие точек, хранение подписи MLSAG потребует $((v+1)\cdot(m+1)+1)\cdot 32 байта$, где v является уровнем смешивания, а m количеством входов. Другими словами, транзакция с 1 входом и общим размером кольца 32 займёт (32 · 2 + 1) · 32 = 2080 байт.

К этому значению добавим 32 байта для хранения образа ключа каждого входа, для $m\cdot 32 байта$, а также дополнительное место для хранения смещённых членов кольца в блокчейне (см. Приложение А). Эти офсеты (смещения) используются верификаторами для вычисления ключа и обязательств каждого выхода члена кольца подписи MLSAG в блокчейне и хранятся как целые числа переменной длины, поэтому мы не можем точно количественно определить необходимое пространство.

Доказательства диапазона (выходы)

Из Раздела 3.4, Раздела 4.3 и Раздела 5.6.3 нам известно, что схема подписи Борромео, используемая Monero для доказательства диапазона, имеет форму n-строки

$$\sigma=\left(c_1, r_{0,1}, r_{0,2}, r_{1,1}, \ldots, r_{63,2}\right)$$

Ключи кольца считаются частью кольцевых подписей. Тем не менее, в этом случае необходимо хранить только обязательства C_j, так как есть возможность выведения дубликата ключей кольца $C_j - 2^j H$ (в целях верификации).

С учётом этого условия доказательство диапазона потребует (1 + 64 · 2 + 64)32 = 6176 байт на выход.

5.7 Кольцевые конфиденциальные транзакции типа RCTTypeSimple

В текущей кодовой базе Monero в отношении транзакций более чем с одним входом используется другая схема подписей, известная как RCTTypeSimple.

Основной особенностью этого подхода является то, что вместо подписания сразу всего набора входов как единого целого, отправитель подписывает каждый вход по отдельности.

Среди прочего это означает невозможность использования кем-либо нулевых обязательств, как это делалось в случае с транзакциями типа RCTTypeFull. Публичный ключ zC является нулевым обязательством если и только если отправителю известен соответствующий приватный ключ z. Если суммы сами по себе в сумме не равны нулю, то по причине сложности определения γ таким образом, чтобы $H=\gamma G$ будет невозможно узнать z.

Давайте рассмотрим ситуацию подробнее. Допустим, Элис хочет подписать вход j. Представьте на секунду, что мы могли бы подписать выражение вроде этого:

$$C_j^a - \sum_t C_t^b = x_j G - \sum_t y_t G + \left(a_j - \sum_t b_t\right) H$$

Так как $a_j - \sum_t b_t \neq 0$, Элис придётся решить DLP для $H=\gamma G$, чтобы получить приватный ключ выражения, то есть, проделать то, что мы считаем довольно сложной вычислительной задачей.

5.7.1 Обязательства по сумме

Как уже объяснялось, если выходы отделяются друг от друга, то отправитель не может подписать совокупное нулевое обязательство относительно выходов текущей транзакции. С другой стороны, подписание каждого входа по отдельности подразумевает промежуточный подход. Отправитель мог бы создать новые обязательства по суммам входов и нулевые обязательства для каждого из ранее потраченных выходов. Таким образом, отправитель мог бы доказать, что транзакция использует в качестве входа только выходы прошлых транзакций.

Другими словами, предположим, что потраченными суммами являются $a_1,\ldots,a_m$. Эти суммы были выходами предыдущих транзакций, в которых у них были обязательства

$$C_j^a = x_j G + a_j H$$

Отправитель может создать новые обязательства для тех же сумм, но уже используя другие скрывающие факторы, то есть

$$C_j^{'a} = x_j^{'} G + a_j H$$

Очевидно, что отправителю был известен приватный ключ из разницы между двумя обязательствами:

$C_j^a - C_j^{'a} = \left(x_j - x_j^{'}\right) G$

следовательно, отправитель мог бы использовать это значение в качестве *нулевого обязательства* для каждого входа. Допустим, $\left(x_j - x_j^{'}\right) = z_j$ и назовём $C_j^{'a}$ *псевдо обязательством по выходу*.

Как и в случае с транзакциями типа RCTTypeFull отправитель может включить зашифрованный скрывающий фактор каждого выхода (маску) как y_t и сумму как b_t в транзакцию (см. Раздел 5.6.1), что позволит каждому получателю t расшифровать y_t и b_t, используя совместно используемые секретные данные $r\,K_t^v$.

Перед отправкой транзакции в блокчейн сеть захочет верифицировать сбалансированность транзакции. В случае с транзакциями типа RCTTypeFull это просто, так как схема подписи MLSAG подразумевает, что каждый отправитель подписался приватным ключом с нулевым обязательством.

В случае с транзакциями RCTTypeSimple скрывающие факторы обязательств по входам и выходам выбираются таким образом, что

$$\sum_j x_j - \sum_t y_t = 0$$

В результате получается следующее:

$$\left(\sum_j C_j^{'a} - \sum_t C_t^b\right) - fH = 0$$

К счастью, процесс выбора таких скрывающих факторов прост. В текущей версии Monero все скрывающие факторы являются случайными, за исключением x_m, который просто задаётся как

$$x_m = \sum_t y_t - \sum_{j=1}^{m-1} x_j$$

5.7.2 Подпись

Как уже упоминалось ранее, в случае с транзакциями RCTTypeSimple каждый вход подписывается отдельно. Мы используем ту же схему подписи MLSAG, что и в случае с транзакциями типа RCTTypeFull, но с другими ключами подписи.

Допустим, Элис подписывает вход j. Этот вход тратит предыдущий выход с ключом $K_{\pi,j}^o$, у которого есть обязательство $C_{\pi,j}^a$. Допустим, что $C_{\pi,j}^{'a}$ является новым обязательством для той же суммы, но с другим скрывающим фактором.

Подобно тому, как это происходит при использовании предыдущей схемы, отправитель выбирает v несвязанных выходов и их соответствующие обязательства из блокчейна, чтобы смешать их с реальным выходом j.

$$K^o_{1,j},\ldots,K^o_{\pi-1,j},K^o_{\pi+1,j},\ldots,K^o_{\nu+1,j}$$

$$C_{1,j},\ldots,C_{\pi-1,j},C_{\pi+1,j},\ldots,C_{\nu+1,j}$$

Затем отправитель может подписать его, используя следующее кольцо:

$$R_j=\left\{\left\{K^o_{1,j},\left(C_{1,j}-C'^a_{\pi,j}\right)\right\}\right\},\ \ldots\ \left\{K^o_{\pi,j},\left(C^a_{\pi,j}-C'^a_{\pi,j}\right)\right\},\ \ldots\ \left\{K^o_{\nu+1,j},\left(C_{\nu+1,j}-C'^a_{\pi,j}\right)\right\}$$

Элис будет знать приватные ключи $k^o_{\pi,j}$ для $K^o_{\pi,j}$ и z_j для нулевого обязательства $\left(C^a_{\pi,j}-C'^a_{\pi,j}\right)$. Она сможет подписать вход j, используя подпись MLSAG в R_j. Как было сказано в Разделе 5.6.3, образы ключей для нулевого обязательства $z_j G$ отсутствуют, и, как следствие, в структуре подписи каждого выхода нет соответствующего образа ключа.

Каждый вход транзакций типа RCTTypeSimple подписывается отдельно с применением схемы, описанной в Разделе 5.6.3, но с использованием таких колец, как R_j, как было определено выше.

Преимущество такого раздельного подписания необходимость в размещении ряда реальных входов и нулевых обязательств по тому же индексу π отсутствует, так как они входят в алгоритм MLSAG. Это означает, что даже если один источник входа будет идентифицирован, источники других входов останутся скрытыми.

Сообщение m, подписанное каждым входом, будет в значительной степени таким же, как и в случае с транзакциями типа RCTTypeFull (см. сноску 8), с той лишь разницей, что оно будет включать в себя псевдо обязательства выхода для этих входов. Создаётся только одно сообщение, и каждый вход MLSAG подписывает его.

5.7.3 Требования к занимаемому месту

Подпись MLSAG (входы)

Каждое кольцо R_j содержит $(\nu+1)\cdot 2$ ключей. Благодаря методу сжатия точек, описанному в Разделе 2.3.2, подпись входа σ потребует $(2(\nu+1)+1)\cdot 32$ *байта*. Сверх этого, вместе с образом ключа $\widetilde{K}^o_{\pi,j}$ и псевдо обязательством выхода $C'^a_{\pi,j}$ это в итоге составит $(2(\nu+1)+3)\cdot 32$ *байта* на вход.

Транзакция с 20 входами, использующая кольцо, включающее в себя 32 члена, займёт ((32 · 2 + 3) · 32)20 = 42880 байт.

Для сравнения, если бы мы использовали схему типа RCTTypeFull для той же самой транзакции, подпись MLSAG и образы ключей потребовали бы (32 · 21 + 3) · 32 + 20 · 32 = 22176 байт.

Доказательства диапазона (выходы)

Размер доказательств диапазона остаётся таким же, как и в случае с транзакциями типа RCTTypeSimple. Как нами было вычислено для транзакций RCTTypeFull, для хранения каждого выхода потребуется 6176 байт.

5.8 Краткое описание концепции

Чтобы кратко изложить суть данной главы, мы хотим представить основное содержание транзакции, для простоты понимания организованное в виде концепции. Реальные примеры транзакций приводятся в Приложениях A и B.

- ***Тип***: «0» обозначает Miner Transaction, «1» обозначает RCTTypeFull, а «2» обозначает RCTTypeSimple
- ***Входы***: для каждого входа $j \in 1,\ldots,m$, потраченного автором транзакции
  - *Смещённые члены кольца*: список «офсетов», указывающий, где верификатор может найти членов кольца $i \in 1,\ldots,v+1$ входа j в блокчейне (включающем реальных вход)
  - *Подпись MLSAG*: члены σ c_1, $r_{i,j}$ и $r_{i,j}^z$ для $i \in 1,\ldots,v+1$ и входа j
  - *Образ ключа*: образ ключа $\widetilde{K}_j^{o,a}$ для входа j
  - *Псевдо обязательство выхода* [только для транзакций типа RCTTypeSimple]: $C_j^{'a}$ для входа j
- ***Выходы***: для каждого выхода $t \in 1,\ldots,p$, направленного на адрес или подадреса $\left(K_t^v, K_t^s\right)$
  - *Одноразовый (скрытый) адрес*: $K_t^{o,b}$
  - *Обязательство по выходу*: C_t^bдля выхода t
  - *Составляющие Диффи-Хеллмана*: чтобы у получателей была возможность вычислить C_t^b и b_t для выхода t
    - *Маска*: $y_t + H_n\left(r\,K_t^v, t\right)$
    - *Сумма*: $b_t + H_n\left(H_n\left(r\,K_t^v, t\right)\right)$
  - *Доказательство диапазона*: для выхода t, использующего кольцевую подпись Борромео
    - *Подпись*: члены σ c_1 и $r_{i,j}$ для $i \in 0,\ldots,63$ и $j \in 1,2$
    - *Побитовые обязательства*: C_i для $i \in 0,\ldots,63$
- ***Комиссия за проведение транзакции***: сообщается простым текстом как умноженная на 10^{12}, поэтому комиссия в размере 1,0 будет записана как 1000000000000
- ***Дополнительные данные***: включают в себя публичный ключ транзакции rG или, если по крайней мере один выход будет направлен на подадрес, $r_t K_t^{s,i}$ для каждого выхода t, направленного на подадрес, и $r_t G$ для каждого выхода t, направленного на обычный адрес, ID платежей и зашифрованные ID платежей

Глава 6

6.0 Блокчейн

Век Интернета расширил границы человеческого опыта. Теперь мы можем общаться с человеком, находящимся в любом уголке мира, и имеем просто невероятный объём информации буквально на кончиках наших пальцев. Обмен товарами и услугами является одной из фундаментальных основ создания мирного и процветающего общества, а в цифровом мире мы можем предложить свои возможности всему миру.

Среда обмена (деньгами) очень важна и является для нас отправной точкой к великому разнообразию экономических благ, которые в противном случае было бы невозможно оценить, а также она позволяет людям, не имеющим ничего общего, взаимодействовать на обоюдовыгодной основе. В течение всей истории человек использовал самые разные виды денег: от морских ракушек до бумажных денег и золота. Тогда деньги передавали из рук в руки, а сейчас ими можно обмениваться при помощи электронных средств.

В наши дни самый распространённый вид, модель электронных транзакций предполагает их обработку сторонними финансовыми организациями. Эти организации принимают деньги на ответственное хранение и им доверяют перевод этих денег по запросу. Такие организации могут выступать в качестве посредника при разрешении споров, проводимые ими платежи обратимы, и они проверяются или контролируются более полномочными организациями.[49]

Век Интернета предполагает возможность создания собственных уникальных валют.

Примечание: эта глава содержит более подробную информацию по реализации, чем предыдущие, так как природа блокчейна сильно зависит от его параметров и специфики структуры.

6.1 Цифровая валюта

Давайте попытаемся и создадим цифровую валюту с нуля.

Допустим, двум приятелям, Джиму и Дуайту, нужна собственная валюта для своего «Секретного общества тайных сыщиков». Джим отправляет Дуайту письмо по электронной почте, где сказано: «Как соучредитель я создаю 5 Стелсбаксов для себя и 5 Стелсбаксов для тебя». Чуть позже, но в тот же день Дуайт обнаруживает, что Кевин создал 69 Стелсбаксов для себя, чтобы купить куклу, китайского болванчика, Дуайта у Джима. Но ведь должно же быть всего 10 Стелсбаксов. Так что же пошло не так?

В случае с ***«email-моделью»*** любой может создать Стелсбаксы, и любой может отправлять свои Стелсбаксы снова и снова. Ресурс не ограничен, а «доказательство двойной траты» отсутствует.

Тогда Джим предлагает новую систему: Стелсбакс 2.0. В этот раз он станет записывать на своём компьютере у кого и сколько Стелсбаксов. Теперь чтобы обменяться ими, людям придётся поговорить с Джимом. Так как Джим является единственным хранителем, то никаких сомнений в том, что Стелсбаксы, которые находятся на компьютере Кевина, являются фальшивыми, уже не возникает. Джим обещает Дуайту, что они начнут со 100 Стелсбаксов у каждого и всё — не больше. Дуайт пытается выкупить свою куклу обратно, но Джим говорит: «Ты должен подтвердить свою личность!» Что снова пошло не так?

В случае с ***«видео игровой моделью»***, когда вся криптовалюта хранится в одной центральной базе данных, пользователи надеются на честность хранителя. Наблюдатели уже не могут проверить поступление Стелсбаксов, а хранитель может изменить правила в любой момент, или же его могут проверить сторонние лица, наделённые очень весомыми полномочиями.

6.1.1 Совместно используемая версия событий

Так как Джиму нельзя доверять управление Стелсбаксами, Дуайт предлагает организовать группу компьютеров, в которой на каждом компьютере хранилась бы запись каждой транзакции, проводимой с использованием Стелсбаксов. И если бы транзакция проводилась на

одном компьютере, то о ней должно было бы сообщаться всем остальным компьютерам, которые приняли бы её только в случае соответствия ряду правил.

Правило 1. Деньги могут быть созданы только по чётко определённому сценарию.

Правило 2. При проведении транзакций могут тратиться только те деньги, которые уже существуют.

Правило 3. При проведении транзакций может быть выведено только такое количество денег, которое будет потрачено.

Правило 4. Транзакции должны форматироваться надлежащим образом.

Правило 5. Только то лицо, у которого находится денежная единица, может потратить её.

Правило 6. Такое лицо может потратить денежную единицу лишь один раз.Правила 2-6 уже охвачены схемами транзакций, описанными в Главе 5 и обеспечивающими преимущества неоднозначности подписи, анонимности получения средств, а также невозможности прочтения передаваемой суммы. О Правиле 1 мы ещё поговорим ниже в этой главе.

А что произойдёт, если один из компьютеров начнёт жульничать и создавать Стелсбаксы для самого себя? Пользователи только получают выгоду от Стелсбаксов, когда другие пользователи принимают их в обмен, поэтому ни один пользователь не примет транзакций с поддельными Стелсбаксами, если возникнет подозрение, что *другие* пользователи просто хотят *узаконить* монеты. Каждый пользователь должен быть честным и следовать тем же правилам, что и остальные пользователи, если хочет потратить свои деньги. Мы называем это «социальным минимумом», «точкой Шеллинга» или «общественным договором».

Но даже простое сохранение данных транзакций вызывает определённую проблему. Если два компьютера получают данные законных транзакций с тратой одних и тех же денег, то до того, как они пошлют информацию друг другу, как им решить, какая информация будет правильной? В случае с криптовалютами вводится понятие «форк» (*fork*), так как возможно наличие двух различных копий, следующих одним и тем же правилам.

Сначала кажется очевидным: более ранняя законная транзакция с тратой денежной единицы и должна считаться «канонической», правильной. Но это проще сказать, чем сделать. Как мы увидим далее, достижение консенсуса в историях транзакций составляет raisond'être (*суть и смысл*) блокчейн технологии.

6.1.2 Простой блокчейн

Для начала нам нужно, чтобы все компьютеры, которые в дальнейшем мы будем называть *узлами*, были согласованы относительно порядка проведения транзакций.

Допустим, Стелсбаксы будут создаваться по объявлению о «генезисе» (*т.е. создании*), которое сделают Джим и Дуайт: «Да будут Стелсбаксы!» Мы называем это сообщение «блоком», и хеш этого блока будет следующим:

$$BH_G = H \text{ (Да будут Стелсбаксы!)}$$

Всякий раз, когда узел принимает какие-либо транзакции, они будут использовать такие хеши транзакций, TH, в качестве сообщений, а также хеш предыдущего блока и вычислять хеши нового блока.

$$BH_1 = H\left(BH_G, TH_1, TH_2, \ldots\right)$$
$$BH_2 = H\left(BH_1, TH_1, TH_2, \ldots\right)$$

И так далее, публикуя каждый новый блок сообщений по мере их создания. Каждый новый блок ссылается на предыдущий, самый последний из опубликованных блоков. Таким образом, чёткий порядок событий распространяется на всю цепочку вплоть до «генезис-сообщения». То есть, у нас получается цепочка блоков: очень простой «блокчейн».[48]

48 Технически блокчейн является направленным ациклическим графом (DAG), при этом, блокчейны типа Bitcoin являются его одномерным вариантом. Графы DAG содержат конечное количество узлов и

Блоки узлов смогут содержать временные метки, которые делают их запись проще. Если большинство узлов имеет верные временные метки, то блокчейн обеспечивает последовательную картину времени записи транзакций.

А что будет, если различные транзакции с одинаковой тратой одних и тех же денег будут добавлены в блоки, ссылающиеся на один и тот же предыдущий блок и опубликованные в одно и то же время? Сеть узлов снова разойдётся (*сделает форк*), так как каждый блок получает один из новых блоков перед другим (для простоты, представьте себе, что у около половины узлов есть каждая сторона форка). Теперь давайте улучшим наш блокчейн.

6.2 Сложность

Если узлы могут публиковать новые блоки всякий раз, когда им вздумается, то сеть, скорее всего, разобьётся и разойдётся на множество различных в равной степени законных цепочек. Скажем, сейчас занимает 30 секунд, чтобы наверняка каждый участник сети получил новый блок. Что было бы, если бы транзакции проводились каждую 31 секунду? Новые блоки просто появлялись бы повсюду непосредственно перед тем, как будет отправлен очередной.

А теперь представьте, что блоки появляются каждые 15 секунд, 10 секунд и так далее? Так как время передачи сообщения является функцией расстояния, сеть разбилась бы на небольшие фракции, обусловленные временем между прохождением одного блока и появлением нового.

Мы можем избежать этого, если будем контролировать скорость создания новых блоков сетью. Если время, необходимое для создания нового блока будет значительно выше времени, за которое предыдущий блок достигнет каждого узла, то сеть останется целой.

6.2.1 Майнинг блока

Выход криптографической хеш-функции распределяется равномерно. Это означает, что любой заданный вход, его хеш, в равной степени вероятно станет каждым отдельно взятым возможным выходом. Кроме того, на вычисление отдельного хеша уходит определённое количество времени.

Давайте возьмём, к примеру, хеш-функцию $H_i(x)$, выходы которой являются числами в пределах от 1 до 100: $H_i(x) \in {}^{D}_{R}\{1,\ldots,100\}$. Мы используем $\in {}^{D}_{R}$, чтобы показать, что выход является детерминированно случайным. При некотором заданном x функция $H_i(x)$ выбирает то же самое «случайное» число от $\{1,\ldots,100\}$ всякий раз, когда вы вычисляете её. На вычисление $H_i(x)$ уходит одна минута.

Допустим, у нас есть сообщение m и так называемый числовой параметр «нонс» (*nonce*) $n=1$, и нам необходимо найти такое значение n, чтобы выходы $H_i(m,n)$ были числом меньше или равны *заданному значению*(*target*)$t=10$ (то есть, $H_i(m,n) \in \{1,\ldots,10\}$). Приблизительный подсчёт и проверка увеличения значения n для каждого нового хеша — сколько примерно хешей это займёт?

И вот хороший ответ: получится приблизительно 10 хешей за 10 минут хеширования, так как шанс, что любой заданный вход окажется выходом, составляет всего 1 к 10. То есть, мы не хотим сказать, что $n=10$ будет правильным ответом. Просто, если мы возьмём множество сообщений и применим этот процесс к каждому из них, то $n=10$ станет *средним* значением.

Мы называем поиск подходящего нонса *майнингом*, а публикация сообщения с его нонсом является *доказательством работы*, так как это доказывает, что мы искали подходящий нонс (даже если нам повезло, и для этого понадобился всего один хеш), и это может проверить каждый, вычислив $H_i(m,n)$.

Теперь, допустим, у нас есть хеш-функция для создания доказательства работы $H_{PoW} \in {}^{D}_{R}\{0,\ldots,m\}$, где m является её максимально возможным выходом. Имея сообщение m (блок информации), нонс n для майнинга, целевое значение t, мы можем вычислить *сложность* d или ожидаемое количество хешей следующим образом: $d=m/t$. Если $H_{PoW}(m,n)*d \le m$, то n принимается.

однонаправленные рёбра (векторы), соединяющие узлы. Если вы начнёте с одного узла, то уже никогда не вернётесь обратно к нему, независимо от того, какое направление будет выбрано. [5]

По мере того, как целевое значение становится меньше, растёт сложность, и компьютеру требуется всё больше и больше хешей, а также всё больше и больше времени, чтобы найти подходящие нонсы.

6.2.2 Скорость майнинга

Допустим, что все узлы одновременно занимаются майнингом нонсов, но выходят из своего «текущего» блока, как только получают новый из сети. Они немедленно начинают майнинг свежего блока, который ссылается на новый.

Предположим, мы собрали группу блоков b из блокчейна (допустим, с индексом $u \in \{1,\ldots,b\}$), каждый из которых имеет значение сложности d_u. Теперь предположим, что узлы, которые произвели майнинг блоков, являются честными, поэтому временная метка каждого блока TS_u является точной. Общее время между самым первым блоком и самым последним блоком будет выражено как $totalTime = TS_b - TS_1$. Приблизительное количество хешей, необходимое для майнинга всех блоков: $total\,Difficulty = \sum_u d_u$.

Теперь мы можем понять, насколько быстро сеть, используя все свои узлы, может вычислять хеши. Если фактическая скорость не изменилась значительно в то время, когда производилась группа блоков, то она должна быть следующей:[49]

$$hashSpeed \approx total\,Difficulty / totalTime$$

Если мы хотим задать целевое время для майнинга новых блоков, чтобы блоки производились с частотой (один блок) / (целевое время), то исходя из скорости хеширования мы можем вычислить, сколько хешей потребуется сети, чтобы потратить такое количество времени на майнинг:

$$miningHashes = hashSpeed * targetTime$$

Так как сложность примерно равна тому, сколько хешей требуется для генерирования доказательства работы, мы может задать новое значение сложности как равное *miningHashes*. Примечание: мы округляем значения, поэтому сложность никогда не будет нулевой.

$$newDifficulty = (total\,Difficulty / totalTime) * targetTime$$

Нет никакой гарантии, что майнинг нового блока займёт то количество хешей, которое соответствует *newDifficulty*, но со временем и с ростом количества блоков, а также при постоянной перекалибровке сложность можно будет использовать для отслеживания реальной хеш-скорости сети, а блоки будут генерироваться близко к *targetTime*.[50]

6.2.3 Консенсус: самая большая совокупная сложность

Теперь у нас есть механизм разрешения конфликтов между форками блокчейна. Так как сложность демонстрирует, какая работа была проделана для майнинга блока, то более высокое значение сложности будет означать, что был проделан больший объём работы.

---

49 Если узел 1 пытается найти нонс n = 23, а позже узел 2 также пытается найти нонс n = 23, то попытка узла 2 будет потрачена впустую, так как сеть уже «знает», что n = 23 не работает (в противном случае узел 1 уже опубликовал бы этот блок). *Фактический* хешрейт сети зависит от того, насколько быстро она хеширует уникальные нонсы для заданного блока сообщений. Как мы увидим далее, поскольку майнеры включают в свои блоки майнинговую транзакцию с одноразовым адресом $K^o \in {}_R^E Z_l$ (где E = effectively, т.е. *фактически*), у майнеров всегда будут уникальные блоки, за исключением ничтожно малого количества случаев.

50 Если мы допустим постоянное постепенное повышение хешрейта сети, то так как сложность будет зависеть от *последних* хешей (то есть, предшествующих даже малейшему повышению хешрейта), то можно ожидать фактическое время генерирования блоков, в среднем, будет чуть меньше, чем $targetTime$. В результате этого график эмиссии (см. Раздел 6.3.1) может быть уравновешен штрафами, связанными с увеличением размеров блоков, о чём будет говориться в Разделе 6.3.2.

По определению, блокчейн с самой высокой совокупной сложностью (всех блоков блокчейна), а следовательно, на построение которого пришёлся наибольший объём работы, считается реальной, законной версией. Если блокчейн разбивается, и каждый форк имеет одну и ту же совокупную сложность, узлы продолжают майнинг в своём форке до тех пор, пока одна ветвь не перегонит другую, и именно в этой точке более слабая ветвь станет заброшенной (то есть, *orphaned*, «осиротевшей»).

Если узлы захотят изменить или обновить базовый протокол, то есть, набор правил, которым следуют узлы при принятии решения, является или нет копия блокчейна или новый блок законными, то они легко могут сделать это, реализовав форк блокчейна. Повлияет или нет новая ветвь каким-либо образом на пользователей, зависит от количества переключившихся узлов, а также от объёма изменений в программной инфраструктуре.[51]

Если злоумышленник захочет убедить честные узлы изменить историю транзакций, возможно, чтобы повторно потратить / отменить трату средств, ему придётся реализовать форк блокчейна (по текущему протоколу), который будет иметь более высокий уровень сложности, чем у текущего блокчейна (который в это время будет расти). Это будет очень трубно сделать, если вы не контролируете 50% хешрейта сети и не можете обогнать другие майнеры. [49]

6.2.4 Майнинг Monero

Чтобы убедиться в том, что форки блокчейна делаются на ровном основании, нам не понадобится отбирать самые последние блоки (для вычисления новых значений сложности). Вместо этого будет необходимо задержать нашу группу b на l. Например, если в блокчейне есть 29 блоков (блоки 1, …, 29), $b=10$, а $l=5$, нам нужно будет отобрать блоки 15-24, чтобы вычислить сложность блока 30.

Если узлы, осуществляющие майнинг, не являются честными, они смогут манипулировать временными метками таким образом, что значения сложности не будут соответствовать реальной скорости хеширования сети. Эту проблему можно решить, рассортировав временные метки в хронологическом порядке, затем отбросив первые посторонние значения o, а после последние посторонние значения o. Теперь у нас есть «окно» блоков $w=b-2*o$. Исходя из предыдущего примера, если $o=3$, а временные метки являются честными, то мы можем отбросить блоки 15-17 и 22-24, оставив блоки 18-21 для вычисления сложности блока 30 на их основе.

Monero является некоторым образом причудливой валютой. Вместо сортировки значений сложности блоков таким образом, чтобы они соответствовали своим отсортированным временным меткам, мы используем совокупные значения сложности множества блоков оригинальной группы, оставляем их не рассортированными, и отбрасываем посторонние значения o. Совокупным значением сложности блока является значение сложности блока плюс значения сложности всех предшествующих блоков в блокчейне.

Используя отброшенные множества временных меток w и совокупные значения сложности (с индексами от 1, …, w), мы можем определить следующее:

$$totalTime = choppedSortedTimestamps[w] - choppedSortedTimestamps[1]$$
$$total\,Difficulty = choppedCumulativeDifficulties[w] - chopped\,CumulativeDifficulties[1]$$

В случае с Monero целевое значение времени составляет 120 секунд (2 минуты), l = 15 (30 минут), b = 720 (один день), а o = 60 (2 часа).[52]

Значения сложности блоков не сохраняются в блокчейне, поэтому кто-либо загружающий копию блокчейна и верифицирующий все блоки на предмет их законности, должен пересчитать все значения сложности на основе записанных временных меток. Существует несколько правил, которые необходимо учитывать при вычислении первых $b+l=735$ блоков.

---

51 Monero успешно изменяла свой протокол 7 раз, при этом, всякий раз практически все пользователи и майнеры принимали изменения. v1 – 18 апреля 2014 [63]; v2 – март 2016; v3 – сентябрь 2016; v4 – январь 2017; v5 –апрель 2017; v6 – сентябрь 2017; v7 –апрель 2018; см. src/cryptonote core/blockchain.cpp mainnet hard forks

52 В марте 2016 (версия 2 протокола) целевое значение времени генерации блоков увеличилось с 1 минуты до 2 минут [11]. Другие параметры сложности всегда оставались такими же.

Правило 1. Следует полностью игнорировать генезис-блок (блок 0 с $d = 1$). У блоков 1 и 2 значение $d = 1$.

Правило 2. Перед тем, как отбросить посторонние значения, необходимо попытаться получить окно w, чтобы вычислить на его основе общие значения.

Правило 3. После блоков w следует отбросить высокие и низкие посторонние значения, приводя отброшенную сумму до b блоков. Если сумма предшествующих блоков (минус w) будет нечётной, следует удалить один или несколько низких, а не высоких посторонних значений.

Правило 4. После b блоков следует отобрать самые ранние b блоки вплоть до $b+1$ блоков, после чего всё пойдёт нормально с задержкой на l.

Доказательство работы Monero

Monero использует хеш-алгоритм доказательства работы (Proof of Work – PoW), известный как Cryptonight, и разработанный таким образом, чтобы быть относительно неэффективным применительно к архитектурам GPU, FPGA и ASIC [60], если сравнивать его работу с работой стандартных хеш-функций, таких как SHA256. В апреле 2017 (версия 7 протокола) этот алгоритм был немного изменён, чтобы сделать невозможным майнинг при помощи микросхем ASIC, разработанных под Cryptonight [22].

6.3 Денежная масса

Очевидно, что цифровой валюте необходима некоторая денежная масса, используя которую пользователи смогли бы совершать транзакции. В случае с валютами, основанными на блокчейн технологии, также известными как криптовалюты, существует два основных механизма создания денег.

Используя первый способ, создатели валюты могут просто задать определённую сумму и распространять её среди людей посредством генезис-сообщения. Часто такой способ называют «эйрдроп» (*airdrop*). Иногда создатели криптовалют создают для себя большую сумму денег путём так называемого «премайнинга» (*pre-mine*) [17].

Если использовать второй способ, то криптовалюта автоматически распределяется в качестве вознаграждения за майнинг блоков, во многом подобно тому, как это происходит при добыче золота. В данном случае существует два типа метода. Модель Bitcoin предполагает наличие «потолка» возможной денежной массы. Награды за блок постепенно сводятся к нулю, и после этой точки уже не будет создано ни одной монеты. При реализации инфляционной модели денежная масса продолжает расти неопределённым образом.

Некоторые криптовалюты используют оба механизма создания денег. Фактически, Monero основана на валюте известной как Bytecoin, у которой был большой премайнинг, за которым последовали вознаграждения за майнинг блоков [10]. У Monero не было премайнинга, и как мы увидим, награды за блок медленно сводятся к небольшой сумме, после которой вознаграждение за все новые блоки будет одинаковым, что делает Monero инфляционной валютой.

6.3.1 Вознаграждение за майнинг блока

Концепция вознаграждения за майнинг блока является предельно простой. Майнеры блоков перед тем, как начать майнинг нонса совершают «майнинговую транзакцию» (*miner transaction*) без входов и одним выходом. Сумма выхода равна вознаграждению за блок плюс комиссии за транзакцию со всех транзакций, входящих в блок, и эта сума указывается простым текстом. Узлы, получающие таким образом добытый блок, должны верифицировать, является ли вознаграждение правильным, и могут вычислить текущую денежную массу путём суммирования всех последних вознаграждений за майнинг блоков вместе.

Помимо того, что при помощи вознаграждений происходит распределение денег, они также стимулируют сам процесс майнинга. Если бы не было таких вознаграждений (или какого-либо другого механизма), зачем тогда кому-либо понадобилось бы майнить блоки? Возможно, только из каких-либо альтруистических побуждений или из простого любопытства. Тем не

менее, наличие нескольких майнеров позволяет злоумышленникам собрать >50% хешрейта сети, используя которые они могут с лёгкостью переписать последнюю историю блокчейна.[53]

При наличии вознаграждение за майнинг блока между манерами происходит соревнование, что поднимает общий хешрейт до тех пор, пока маргинальная стоимость добавления хешрейта не станет выше, чем маргинальное вознаграждение за получение пропорции добытых блоков (которая становится постоянным коэффициентом) (плюс некоторые премиальные за риск или альтернативные издержки). Это означает, что валюта становится более ценной, её общий хешрейт растёт, и становится всё более сложно и затратно добыть >50%.

Сдвиг разрядов

Сдвиг разрядов используется для вычисления базового вознаграждения за майнинг блока (как будет сказано в Разделе 6.3.2, вознаграждение иногда может опуститься ниже базовой суммы).

Предположим, у нас есть целое число A = 13 в двоичном представлении [1101]. Если сдвинуть разряды A вниз на 2, что можно обозначить как A >> 2, то мы получим [0011].01, что равняется 3,25. В реальности эти последние .01 находятся за пределами множества, будучи равными 0,25, и они отбрасываются, «сдвигаются» в никуда, и в результате у нас остаётся [0011] = 3.

Эта операция эквивалентна $floor(13/4)=3$ или $floor(A/2^2)$, где *floor* округляет число вниз, отбрасывая часть 01 из [0011].01.

Вычисление базового вознаграждения за блок Monero

Обозначим существующую совокупную денежную массу как M, а «предел» денежной массы как $L = 2^{64} - 1$ (в двоичном представлении это будет выглядеть как [11….11], с 64 разрядами). В самом начале базовое вознаграждение за блок Monero составляло B = (L-M) >> 20, или же, другими словами, $floor((L-M)/2^2)$. Если M = 0, то в десятичном формате мы получим следующие значения:

$$L = 18, 446, 744, 073, 709, 551, 615$$
$$B_0 = (L - 0) >> 20 = 17, 592, 186, 044, 415$$

Эти числа являются «атомными единицами» — 1 атомную единицу Monero нельзя поделить (не бывает атомных единиц равных 0,5). Очевидно, что атомные единицы просто смехотворны — значение L составляет более 18 квинтиллионов! Мы можем поделить всё на 10^12, чтобы подвинуть десятичную запятую и получить стандартные единицы Monero (также известные как XMR, так называемый «биржевой тикер» Monero).

$$\frac{L}{10^{12}} = 18,446,744,073709551615$$
$$B_0 = \frac{(L-0) \gg 20}{10^{12}} = 17,592186044415$$

И так мы получаем вознаграждение за самый первый блок, которое, к слову, ушло человеку под псевдонимом thankful_for_today (который начал проект Monero) в генезис-блоке [63] и составило 17,6 Monero! Убедиться в этом вы можете, ознакомившись с Приложением D.[54]

По мере майнинга всё большего количества блоков аккумулируется вознаграждение за блок, и M растёт, что непрерывно снижает вознаграждение за будущие блоки. Изначально (с появлением генезис-блока в апреле 2014) блоки Monero вычислялись по одному в минуту, но уже в марте 2016 майнинг одного блока занимал две минуты [11]. В целях соблюдения и сохранения «графика эмиссии», то есть, частоты создания денег,[55] вознаграждения за майнинг блока были удвоены. Это просто означало, что после изменения для новых блоков мы стали

53 Если злоумышленник получает большую долю (свыше 50%) хешрейта, ему становится всё проще и проще переписывать историю всё более старых блоков.

54 Суммы Monero сохраняются в формате атомных единиц в блокчейне.

55 Интересное сравнение графиков эмиссии Monero и Bitcoin приводится в работе [15].

использовать (L-M) >> 19 вместо >> 20. В настоящее время базовое вознаграждение за майнинг блока является следующим:

$$B = \frac{(L - M) \gg 19}{10^{12}}$$

6.3.2 Штраф за размер блока

Было бы прекрасно, если бы каждая новая майнинг транзакция превращалась в блок. Тем не менее, а что будет, если кто-то проведёт множество транзакций со злым умыслом? Блокчейн, в котором хранятся данные каждой транзакции, быстро вырастет просто до неприличных размеров.

Один из способов избежать этого заключается в ограничении размера блока, то есть, когда количество транзакций в блоке будет фиксированным. А что будет, если объём честной транзакции вырастет? Автор каждой транзакции будет вести борьбу за место в новом блоке, предлагая комиссии майнерам. Майнеры будут выбирать транзакции с самыми высокими комиссиями, чтобы заработать как можно больше денег. По мере роста объёма транзакций комиссии могут стать слишком большими относительно транзакций, в ходе которых проводятся малые суммы (как в случае, когда Элис покупает яблоко у Боба). В блокчейн будут попадать только транзакции тех людей, которые хотят обойти всех.

Monero позволяет избежать таких крайностей (установки ограниченного и безлимитного размера блока), так как предполагает использование динамичного размера блока. Майнеры могут создавать блоки, размер которых будет больше, чем у стандартных, но в этом случае им придётся заплатить штраф, который будет заключаться в снижении вознаграждения за блок. Это цена создания большого блока.

Чтобы вычислить сумму штрафа за размер блока, нам понадобится взять последние 100 блоков в блокчейне и найти *среднее значение* размера блока, *median_100blocks*. Мы задаём переменной M100 большее значение в ряду {*median_100blocks*, 300kB}.[56] Только на блоки со значением выше M100 налагается штраф. Тем не менее, среднее значение может расти, что постепенно позволит создавать блоки большего размера без каких-либо штрафов. Максимальный размер блока составляет 2*M100.

Если заданный размер блока больше M100, то, при базовом вознаграждении за блок B, штраф, налагаемый на вознаграждение, будет следующим:

$$P = B \text{ ¿ } ((\text{block\_size}/M100) - 1)^2$$

Следовательно, фактическое вознаграждение будет уже таким:

$$B^{actual} = B - P$$
$$B^{actual} = B \text{ ¿ } (1 - ((\text{block\_size}/M100) - 1)^2)$$

Использование операции ^2 означает, что штрафы субпропорциональны размеру блока. В случае, если размер блока на 10% больше M100, штраф составит лишь 1%, и так далее. [15]

Можно ожидать, что майнеры станут создавать блоки, размер которых будет превышать M100, если комиссия за добавление очередной транзакции превысит размер штрафа.

6.3.3 Динамическая минимальная комиссия

Чтобы злоумышленники не смогли заполонить блокчейн своими транзакциями, например, с целью «заражения» кольцевых подписей, и в целом раздуть его без какой-либо необходимости, Monero требуется определить минимальный размер комиссии на kB данных

56 В начале после появления Monero это значение составляло 20 kB, затем, в марте 2016 (версия 2 протокола) [11], оно выросло до 60 kB, а с апреля 2017 (версия 5 протокола) [1], оно составляет уже 300 kB. Такой не нулевой минимальный уровень динамического размера блока обеспечивает возможность временного изменения объёма транзакций. Это особо помогало на ранних этапах распространения Monero.

транзакции. Изначально комиссия просто составляла 0,002 XMR/kB. Однако, в январе 2017 (версия 4 протокола) была добавлена формула, которая позволила избежать некоторых рисков, связанных с безопасностью и с превышением размера комиссии суммы вознаграждения за блок, а также демотивировать майнеров от превышения M100 значения 300 kB при прохождении транзакций больших объёмов [8].

Сначала мы установим базовую динамическую комиссию как $f_b^{kB}=0{,}000/\text{kB}$,[57] а затем вычислим B^{actual} и M100, как это было описано в предыдущих разделах.

Минимальная комиссия на kB составит[58]

$$f^{kB}=f_b^{kB}*(300\,kB/M\,100)*(B^{actual}/10)$$

Примечание: мы округляем размер транзакции до ближайшего kB.

6.3.4 Дополнительная эмиссия

Предположим, что есть некая криптовалюта с фиксированным максимальным количеством денежной массы и динамическим размером блока. Спустя какое-то время вознаграждение за блок достигнет нулевого значения. Более не будет никаких штрафов за превышение размера блока, майнеры смогут просто добавлять любую транзакцию с ненулевой комиссией в свои блоки.

Размеры блоков в среднем стабилизируются для всех транзакций, попадающих в сеть, а у авторов транзакций более не будет причины соревноваться, используя комиссии за транзакции сверх минимума, которые станет нулевым согласно Разделу 6.3.3.

Это создаёт нестабильную и небезопасную ситуацию. У майнеров практически не будет или не будет совсем никакой мотивации заниматься майнингом новых блоков, что приведёт к падению хешрейта сети и обернётся падением инвестиций. Время вычисления блоков останется тем же по мере корректировки сложности, но в результате риск проведения злоумышленниками атак путём двойной траты станет более ощутимым.

В случае Monero эта проблема решена следующим образом: вознаграждение за блок не может упасть ниже 0,6 XMR (0,3 XMR в минуту). Это означает, что если соблюдается следующее условие

$$0{,}6 > ((L - M) >> 19)/10_{12}$$
$$M > L - 0{,}6 ¿ 2^{19} ¿ 10^{12}$$
$$M/10^{12} > L/10^{12} - 0{,}6 ¿ 2^{19}$$
$$M/10^{12} > 18,\,132,\,171{,}273709551615$$

то блокчейн Monero никогда не войдёт в состояние так называемой «дополнительной эмиссии» (*emission tail*) и вознаграждение в размере 0,6 XMR (0,3 XMR в минуту) будет выплачиваться по-прежнему.[59]

6.3.5 Майнинговая транзакция

В каждом блоке есть майнинговая транзакция, которая позволяет любому лицу, который занимался майнингом блока, отправить себе вознаграждение за блок и любые комиссии за транзакции, входящие в блок (суммируются в один выход).[60] Выход майнинговой транзакции

57 В апреле 2017 (версия 5 протокола) [1] размер базовой комиссии сократился с 0,002 XMR/kB до 0,0004 XMR/kB.

58 Чтобы проверить, является ли данная комиссия правильной, мы оставляем 2% буфер для f^{kB} на случай численной избыточности (мы вычисляем комиссию ещё до того, как будет полностью определён размер транзакции). Это означает, что фактический минимум комиссии составит $0{,}98*f^{kB}$.

59 Начало дополнительной эмиссии Monero прогнозируется на май 2022 [13].

60 Выход майнинговой транзакции теоретически может быть отправлен на подадрес и/или использоваться в мультиподписи, и/или зашифрованный (или нет) ID платежа. Нам не известно, используются или нет эти возможности в каком-либо из вариантов реализации. Следует отметить, что при этом, как обычно, требуется

является заблокированным, то есть, не может быть потрачен в течение 60 блоков после того, как был опубликован [18].[61]

С тех пор, как в январе 2017 (версия 4 протокола) [12] был реализован протокол RingCT, людям, загружающим новую копию блокчейна, приходится вычислять обязательство по сумме a их майнинговой транзакции (также известной как tx) как $C = 1G + aH$, и сохранять его для ссылки. Это означает, что майнеры блоков могут тратить выходы своих майнинговых транзакций как и выходы обычных транзакций, смешивая их с другими выходами в кольцах MLSAG (как обычных, так и майнерских транзакций).

После введения RingCT верификаторы блокчейна сохраняют обязательство по суме каждой майнинговой транзакции блока, каждое из которых составляет 32 байта, и вычисляют их, используя:

PA Добавление точки (*Point Addition*) для некоторого количества точек A, B: $A + B$ [1]

KBSM Скалярное умножение с известной основой (*Known-Base Scalar Multiplications*) для некоторого целого числа a: aG [2]

6.4 Структура блокчейна

Monero использует простой стиль блокчейна.

Блокчейн начинается с генезис-сообщения какого-либо рода (в нашем случае это, как правило, майнинговая транзакция, распределяющая вознаграждение за первый блок), создающего генезис-блок. Следующий блок содержит ссылку на предыдущий блок в форме ID (*идентификатора*) блока. ID блока является простым хешем заголовка блока (списка данных блока), так называемого «корня Меркла», который присоединяет все ID транзакций блока (которые являются хешами каждой транзакции), и количества транзакций (включая майнинговую транзакцию).

Чтобы создать новый блок, необходимо произвести хеши доказательства работы, изменяя значение нонса, которое хранится в заголовке блока, до тех пор, пока не будет выполнено целевое условие сложности. Доказательство работы и ID блока хешируют одну и ту же информацию, но используют разные хеш-функции.

6.4.1 ID транзакции

Идентификаторы транзакций похожи на сообщения, подписанные входом подписей MLSAG (см. Раздел 5.6.3), но также включают в себя подписи MLSAG.

Хешируется следующая информация:

2 Префикс транзакции = {era-версия транзакции (то есть, RingCT = 2), входы {смещения ключей, образы ключей}, выходы {одноразовые адреса}, дополнительные данные {публичный ключ транзакции, ID платежа или зашифрованный ID платежа, прочее}}.

3 Тело транзакции = {тип подписи (нулевая/майнинговая или простая, или полная), размер комиссии за транзакцию, псевдо обязательства выходов для входов, ecdhInfo (маски и суммы), обязательства по выходам}.

4 Подписи = {подписи MLSAG, доказательства диапазона}.

На следующей диаграмме черными стрелками показаны хеши входов.

TX Prefix	*Префикс транзакции*
TX Stuff	*Тело транзакции*
Signatures	*Подписи*
Transaction ID	*Идентификатор транзакции*

наличие публичного ключа.

61 Автор любой транзакции может заблокировать свои выходы, сделав их невозможными для траты до тех пор, пока блок не достигнет определённой высоты. Он может только заблокировать все выходы до достижения одной и той же высоты блока. Непонятно, имеет ли это какой-то практический смысл для авторов транзакций. Блокировка выходов является обязательной только для майнинговых транзакций.

Вместо «входа» майнинговая транзакция записывает высоту своего блока. Это гарантирует, что ID майнинговой транзакции, который по сути является ID обычной транзакции, за тем лишь исключением, что H_n (Подписи) $\rightarrow H_n(0)$, всегда будет уникальным. Никто не сможет создать других блоков с таким же внутренним идентификатором транзакции, установив ту же сумму майнинговой транзакции (то есть, размер комиссий) и одноразовый адрес выхода, так как это может запутать людей, занимающихся поиском ID.

6.4.2 Дерево Меркла

Некоторые пользователи могут пожелать выбросить ненужные данные из своей копии блокчейна. Например, когда вы проверяете доказательства диапазона и подписи входов некоторых транзакций, единственной причиной сохранения этой информации по подписям является возможность самостоятельной проверки транзакции пользователями, получившими её от вас.

Чтобы облегчить «обрезание» данных транзакций, а также для их более общей организации внутри блока, нами используется дерево Меркла [47], которое является простым двоичным хеш-деревом идентификаторов транзакций. Любая ветвь дерева Меркла может быть обрезана, если вы сохраните её корневой хеш.[62]

На рисунке 6.1 схематически показан пример дерева Меркла, в основе которого лежат четыре обычные и одна майнинговая транзакция.[63]

Figure 6.1: Merkle Tree	*Рисунок 6.1. Дерево Меркла*
Transaction ID	*Идентификатор транзакции*
Miner Transaction ID	*Идентификатор майнинговой транзакции*
Hash	*Хеш*
Merkle Root	*Корень Меркла*

Как видно, корень Меркла ссылается на все транзакции блоков, что облегчает обрезание транзакции или какого-либо компонента транзакции.

6.4.2 Блоки

Блок в основном состоит из заголовка блока и нескольких транзакций. В заголовках блока записывается важная информация о каждом блоке. На транзакции, входящие в состав блока, можно выйти через корень Меркла. Ниже мы кратко описано содержание блоков. Пример реального блока приводится в Приложении С.

5 Заголовок блока

6 ***Старшая версия***. Использовалась для отслеживания хардфорков (изменений протокола).

7 ***Младшая версия***. Ранее использовалась для голосования, а теперь снова используется для отображения старшей версии.

8 ***Временная метка***. Время блока по UTC (всемирное координированное время). Добавляется майнерами. Временные метки не верифицируются, но они не будут приняты, если будут ниже среднего значения всех временных меток последних 60 блоков.

9 ***ID предыдущего блока***. Ссылка на предыдущий блок. Это важная характеристика блокчейна.

62 Нам неизвестен кто-либо из пользователей Monero, кто обрезал бы свой блокчейн, а также какое-либо программное обеспечение, способное сделать это. Если бы обрезание *было* реализовано, вероятнее всего, оно включало бы в себя удаление всех данных подписей после верификации, а также сохранение H_n (Подписей) для вычисления идентификаторов транзакций. Подписи составляют большую часть данных блока, поэтому в результате такого удаления размер блокчейна значительно сократился.

63 Ошибка в коде вычисления дерева Меркла стала причиной серьёзной атаки на Monero 4 сентября 2014 [38].

10 ***Нонс***. 32-байтное целое число, которое майнеры меняют снова и снова до тех пор, пока хеш PoW не будет соответствовать целевому значению сложности. Верификаторы блока легко могут пересчитать хеш PoW.

6 **Майнинговая транзакция**. Распределяет вознаграждение за блок и комиссию за транзакции между майнерами блока.

7 **Идентификаторы (ID) транзакции**. Ссылаются на не майнинговые транзакции, добавленные в блокчейн вместе с этим блоком. ID транзакций могут, в сочетании с ID майнинговой транзакции, использоваться для вычисления корня Меркла, а также для вычисления фактических транзакций, где бы те не хранились.

ID блоков вычисляются следующим образом:[64]

ID блока = H_n (заголовок блока, корень Меркла, количество транзакций + 1)

А майнинг блока происходит следующим образом:

11 пока $PoW_{output} > target$, продолжать изменение нонса и пересчитывать:

$PoW_{output} = H_{PoW}$ (заголовок блока, корень Меркла, количество транзакций + 1)

Помимо данных каждой транзакции (см. Раздел 5.8), мы сохраняем следующую информацию:

8 Старшую и младшую версии: переменные целые числа < 64 бит.

9 Временную метку: переменное целое число < 64 бит.

10 ID предыдущего блока: 32 байта.

11 Нонс: 32 байта.

12 Майнинговую транзакцию: 32 байта на одноразовый адрес, 32 байта на публичный ключ транзакции, а также переменные целые числа для времени разблокировки, высоты и суммы. После загрузки блокчейна нам также понадобится 32 байта для хранения обязательства по суммам майнинговых транзакций после реализации RingCt, $C = 1G + aH$, которые вычисляются при помощи:

PA Добавление точки для некоторого количества точек A, B: $A + B$ [1]

KBSM Скалярное умножение с известной основой для некоторого целого числа a: aG [2]

13 Идентификаторы транзакций: по 32 байта на каждый.

64 +1 означает майнинговую транзакцию.

Библиография

[1] Add intervening v5 fork for increased min block size.

https://github.com/monero-project/monero/pull/1869
[Online; accessed 05/24/2018].

[2] cryptography - what is a cryptographic oracle?

https://security.stackexchange.com/questions/10617/what-is-a-cryptographic-oracle
[Online; accessed 04/22/2018].

[3] Cryptonote address tests.

https://xmr.llcoins.net/addresstests.html
[Online; accessed 04/19/2018].

[4] Devmeeting 2018-05-06.

https://monerobase.com/wiki/DevMeeting_2018-05-06
[Online; accessed 05/06/2018].

[5] Directed acyclic graph.

https://en.wikipedia.org/wiki/Directed_acyclic_graph
[Online; accessed 05/27/2018].

[6] How do payment ids work?

https://monero.stackexchange.com/questions/1910/how-do-payment-ids-work
[Online; accessed 04/21/2018].

[7] How does monero's privacy work?

https://www.monero.how/how-does-monero-privacy-work
[Online; accessed 04/04/2018].

[8] How does the dynamic fee calculation work?

https://monero.stackexchange.com/questions/2531/how-does-the-dynamic-fee-calculation-work
[Online; accessed 05/25/2018].

[9] Modular arithmetic.

https://en.wikipedia.org/wiki/Modular_arithmetic
[Online; accessed 04/16/2018].

[10] Monero inception and history.

https://monero.stackexchange.com/questions/475/

monero-inception-and-history-how-did-monero-get-started-what-are-its-origins-a/476#476

[Online; accessed 05/23/2018].

[11] Monero v0.9.3 - hydrogen helix - released!

https://www.reddit.com/r/Monero/comments/4bgw4z/monero_v093_hydrogen_helix_released_urgent_and/
[Online; accessed 05/24/2018].

[12] Ring ct; moneropedia.

https://getmonero.org/resources/moneropedia/ringCT.html
[Online; accessed 06/05/2018].

[13] Tail emission.

https://getmonero.org/resources/moneropedia/tail-emission.html
[Online; accessed 05/24/2018].

[14] Trust the math? an update.

http://www.math.columbia.edu/~woit/wordpress/?p=6522
[Online; accessed 04/04/2018].

[15] Useful for learning about monero coin emission.

https://www.reddit.com/r/Monero/comments/512kwh/useful_for_learning_about_monero_coin_emission/d78tpgi/
[Online; accessed 05/25/2018].

[16] Varint description; issue #2340.

https://github.com/monero-project/monero/issues/2340#issuecomment-324692291
[Online; accessed 06/14/2018].

[17] What is a premine?

https://www.cryptocompare.com/coins/guides/what-is-a-premine/
[Online; accessed 06/11/2018].

[18] What is the block maturity value seen in many pool interfaces?

https://monero.stackexchange.com/

questions/2251/what-is-the-block-maturity-value-seen-in-many-pool-interfaces
[Online; accessed 05/26/2018].

[19] Xor – from wolfram mathworld.

http://mathworld.wolfram.com/XOR.html
[Online; accessed 04/21/2018].

[20] Nist releases sha-3 cryptographic hash standard, August 2015.

https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard
[Online; accessed 06/02/2018].

[21] Issue with proof of unforgeability of asnl, 2016.

https://github.com/monero-project/research-lab/issues/4
[Online; accessed 04/04/2018].

[22] Monero cryptonight variants, and add one for v7, April 2018.

https://github.com/monero-project/monero/pull/3253
[Online; accessed 05/23/2018].

[23] Prometheus Tereno Albert Werner, Montag. Cryptonote transaction extra field. CryptoNote, October 2012.

https://cryptonote.org/cns/cns005.txt
[Online; accessed 04/04/2018].

[24] Kurt M. Alonso and Jordi Herrera Joancomart´ı. Monero - privacy in the blockchain. Cryptology ePrint Archive,

Report 2018/535, 2018.
https://eprint.iacr.org/2018/535

[25] Adam Back. Ring signature efficiency. BitcoinTalk, 2015.

https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684
[Online; accessed 04/04/2018].

[26] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, Public Key Cryptography - PKC 2006 , pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[27] Daniel J. Bernstein. Chacha, a variant of salsa20, 2008.

[28] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards Curves , pages 389–405. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[29] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. Journal of Cryptographic Engineering , 2(2):77–89, Sep 2012.

[30] Daniel J. Bernstein and Tanja Lange. Faster Addition and Doubling on Elliptic Curves , pages 29–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[31] Karina Bjørnholdt. Dansk politi har knækket bitcoin-koden, May 2017.

http://www.dansk-politi.dk/artikler/2017/maj/dansk-politi-har-knaekket-bitcoin-koden
[Online; accessed 04/04/2018].

[32] David Chaum and Eug`ene Van Heyst. Group signatures. In Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques , EUROCRYPT'91, pages 257–265, Berlin, Heidelberg, 1991. Springer-Verlag.

[33] W. Diffie and M. Hellman. New directions in cryptography. IEEE Trans. Inf. Theor. , 22(6):644–654, September 2006.

[34] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, Advances in Cryptology — CRYPTO' 86 , pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. [35] Eiichiro Fujisaki and Koutarou Suzuki. Traceable Ring Signature , pages 181–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[36] Thomas C Hales. The NSA back door to NIST. Notices of the AMS , 61(2):190–192.

[37] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. Guide to Elliptic Curve Cryptography . Springer- Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[38] Surae Noether Jan Macheta, Sarang Noether and Javier Smooth. Counterfeiting via merkle tree exploits within virtual currencies employing the cryptonote protocol, mrl-0002, September 2014.

https://lab.getmonero.org/pubs/MRL-0002.pdf
[Online; accessed 05/27/2018].

[39] Don Johnson and Alfred Menezes. The elliptic curve digital signature algorithm (ecdsa). Technical Report CORR 99-34, Dept. of C&O, University of Waterloo, Canada, 1999.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.9475&rep=rep1&type=pdf
[Online; accessed 04/04/2018].

[40] Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, January 2017.

[41] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Proceedings, Part II, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9815 , pages 33–61, Berlin, Heidelberg, 2016. Springer-Verlag.

[42] Alexander Klimov. ECC patents?, October 2005.

http://article.gmane.org/gmane.comp.encryption.general/7522
[Online; accessed 04/04/2018].

[43] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups , pages 325–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. [44] Ueli Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, Progress in Cryptology – AFRICACRYPT 2009, pages 272–286, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[45] Greg Maxwell. Confidential transactions.

https://people.xiph.org/~greg/confidential_values.txt
[Online; accessed 04/04/2018].

[46] Gregory Maxwell and Andrew Poelstra. Borromean ring signatures. 2015.

https://pdfs.semanticscholar.org/4160/470c7f6cf05ffc81a98e8fd67fb0c84836ea.pdf [Online; accessed 04/04/2018].

[47] R. C. Merkle. Protocols for public key cryptosystems. In 1980 IEEE Symposium on Security and Privacy, pages 122–122, April 1980.

[48] Victor S Miller. Use of elliptic curves in cryptography. In Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85, pages 417–426, Berlin, Heidelberg, 1986. Springer-Verlag.

[49] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system,"http://bitcoin.org/bitcoin.pdf, 2008.

[50] Arvind Narayanan and Malte M öser. Obfuscation in bitcoin: Techniques and politics. CoRR, abs/1706.05432,2017.

[51] Sarang Noether and Brandon Goodell. An efficient implementation of monero subaddresses, mrl-0006, October 2017.

https://lab.getmonero.org/pubs/MRL-0006.pdf [Online; accessed 04/04/2018].

[52] Shen Noether and Sarang Noether. Monero is not that mysterious, mrl-0003, September 2014.

https://lab.getmonero.org/pubs/MRL-0003.pdf [Online; accessed 06/15/2018].

[53] Michael Padilla. Beating bitcoin bad guys, August 2016.

http://www.sandia.gov/news/publications/labnews/articles/2016/19-08/bitcoin.html [Online; accessed 04/04/2018].

[54] Torben Pryds Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing, pages 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[55] luigi1111 Riccardo "fluffypony" Spagni. Disclosure of a major bug in cryptonote based currencies, May 2017.

https://getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html [Online; accessed 04/10/2018].

[56] Adi Shamir Ronald L. Rivest and Yael Tauman. How to leak a secret. C. Boyd (Ed.): ASIACRYPT 2001, LNCS 2248, pp. 552-565, 2001.

https://people.csail.mit.edu/rivest/pubs/RST01.pdf [Online; accessed 04/04/2018].

[57] SJD AB S. Josefsson and N. Moeller. Eddsa and ed25519. Internet Research Task Force (IRTF), 2015.

https://tools.ietf.org/html/draft-josefsson-eddsa-ed25519-03
[Online; accessed 05/11/2018].

[58] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, Advances in Cryptology — CRYPTO' 89 Proceedings , pages 239–252, New York, NY, 1990. Springer New York.

[59] Paola Scozzafava. Uniform distribution and sum modulo m of independent random variables. Statistics & Probability Letters, 18(4):313 – 314, 1993.

[60] Seigen, Max Jameson, Tuomo Nieminen, Neocortex, and Antonio M. Juarez. Cryptonight hash function. CryptoNote, March 2013.

https://cryptonote.org/cns/cns008.txt
[Online; accessed 04/04/2018].

[61] Adam Mackenzie Shen Noether and Monero Core Team. Ring confidential transactions, mrl-0005, February 2016.

https://lab.getmonero.org/pubs/MRL-0005.pdf
[Online; accessed 06/15/2018].

[62] QingChun ShenTu and Jianping Yu. Research on anonymization and de-anonymization in the bitcoin system. CoRR , abs/1510.07782, 2015.

[63] thankful for today. [ann][bmr] bitmonero - a new coin based on cryptonote technology - launched, April 2014. Monero's actual launch date was April 18th , 2014.

https://bitcointalk.org/index.php?topic=563821.0
[Online; accessed 05/24/2018].

[64] Nicolas van Saberhagen. Cryptonote v2.0.

[Online; accessed 04/04/2018].

[65] A. Langley Google Inc. Y. Nir, Check Point. Chacha20 and poly1305 for ietf protocols. Internet Research Task Force (IRTF), May 2015.

https://tools.ietf.org/html/rfc7539
[Online; accessed 05/11/2018].

Приложения

Приложение А

Структура транзакций RCTTypeFull

В этом приложении мы приводим пример распечатанных данных реальной транзакции Monero типа RCTTypeFull, а также приводим свои пояснения к соответствующим полям.

Данные были выведены при помощи команды print tx <TransactionID> +hex +json в демон-программе monerod в не отключённом режиме. <TransactionID> является хешем транзакции. Первая распечатанная строка показывает фактический запуск команды.

Чтобы воспроизвести наши результаты, пользователи могут сделать следующее:[65]

1. Вам понадобится интерфейс с командной строкой (CLI) Monero, который можно найти на getmonero.org/downloads (и не только). Следует выбрать [Command Line Tools Only] для вашей операционной системы, переместить файл в нужное место и разархивировать его.
2. После этого, следует открыть [terminal/command line] и перейти к папке, созданной в результате разархивирования.
3. Используя ./monerod, открыть monerod. Теперь блокчейн Monero будет загружен. К сожалению, на данный момент простого способа распечатки транзакций не существует, поэтому приходится загружать блокчейн.
4. После того, как процесс синхронизации будет завершён, можно будет использовать команды вроде print_tx. Чтобы узнать другие команды, используйте [help].

В силу характера статьи, мы сократили длинные шестнадцатеричные последовательности, и оставили только их начало и конец: 0200010c7f[...]409.

Компонент rctsig_prunable, как следует из его названия, теоретически можно *вырезать* из блокчейна. То есть, после того, как блок будет согласован, а существующие правила длины блокчейна исключают всяческую возможность атаки, основанной на двойной трате, всё это поле можно будет вырезать и заменить его хешем в дереве Меркла. Так ещё никогда не делалось в блокчейне Monero, но, тем не менее, это вполне возможно, и позволит сэкономить значительное количество места.

Образы ключей и ключи колец хранятся отдельно в той части транзакций, которая не может быть удалена. Эти компоненты важны для обнаружения атак, основанных на двойной трате, и не могут быть вырезаны.

В нашем примере транзакция имеет один вход и два выхода, и была добавлена в блокчейн с временной меткой 2017-12-18 20:28:11 UTC (как было указано майнером блока).

```
print_tx b43a7ac21e1b60ad748ec905d6e03cf3165e5d8c9e1c61c263d328118c42eaa6 +hex +json
Found in blockchain at height 1467685
0200010c7f[...]409
{
      "version": 2,
      "unlock_time": 0,
      "vin": [{
              "key":{
                      "amount": 0,
                      "key_offsets": [ 799048, 782511, 1197717, 216704, 841722
                      ],
                      "k_image": "595a612d0df27181c46a8af70a9bd682f2a000124b873ba5d2b9f4b4e4efd672"
              }
      }
      ],
      "vout": [{
              "amount": 0,
              "target":{
                      "key": "aa9595f55f2cfaed3bd2a67453bb064dc7fd454a09c2418d7338782790185fe3"
              }
      },{
```

65 Данные блокчейна также можно найти в сетевом блок-эксплорере, таком как https://moneroblocks.info/.

```
22              "amount": 0,
23              "target":{
24                      "key": "0ccb48ed2ebbcaa8e8831111029f3300069cff0d1408acffbfc3810b362ea217"
25              }
26      }
27      ],
28      "extra": [ 2, 33, 0, 129, 70, 77, 194, 248, 93, 24, 94, 15, 107, 233, 0, 229, 82,
29      175, 243, 123, 58, 204, 135, 171, 100, 101, 192, 42, 187, 157, 168, 222, 98, 192,
30      110, 1, 1, 185, 87, 22, 38, 116, 81, 124, 85, 68, 36, 44, 229, 235, 46, 159, 139,
31      114, 234, 211, 50, 41, 28, 92, 26, 249, 184, 228, 197, 64, 139, 5
32      ],
33      "rct_signatures":{
34              "type": 1,
35              "txnFee": 26000000000,
36              "ecdhInfo": [{
37                      "mask": "68f508c5515694ce5a33b316b990e8b67a944725c93d806767e61b2e0b13d300",
38                      "amount": "913372a2424b22bd9712183f5a7c8027c8d9af89b52d1e7d06fd1f87a1e5d20d"
39              },{
40                      "mask": "fbc3e5bdb36fc58e5800ffc549ab7bd533fadb7e6b64898c82ea620d749fc80e",
41                      "amount": "b9335c3dc0afb774f812f9f58a412c849f3c828d873f1c16ab102963799d9809"
42              }],
43              "outPk": [ "cf141f5dfe04df14afad6b451d600aa5826a9be44a76a1630850c1d5951d482e",
44                      "e10bb69b66af5dabec765c7f5f7528926088877fa36746833828a0575896ae57"]
45      },
46      "rctsig_prunable":{
47              "rangeSigs": [{
48                      "asig": "b9b544a7[...]d4c5726e81c4c4b6205dacc05208",
49                      "Ci": "bc7ae457[...]fe490458"
50              },{
51                      "asig": "9c457b41[...]545b60c",
52                      "Ci": "ce9b4d8e[...]03a6752"
53              }],
54              "MGs": [{
55                      "ss": [[ "a8120b96f5f2ac5bceab37f7d6bf8d86554d87c4af3441007cad92f54a24d908",
56                      "2e6bc016297a5d398936c9f45e7a80215138f69e55179b337922e2d51c1a9f00"],
57                      ["1e1052a68c38bb88b6e8f257d999c13f1d5f4fa219cc23479ccbfa6b14b5960a",
58                      "e914d35eed0d27344fbc3a89b91bd445d433b561efc844c9f466a61ebb5f6d09"],
59                      ["e04d011f515461fdbd8d13536c23143dc365d87dd323defb1af834e540a8fc0e",
60                      "f9b41a117a1415fec54f1cc16aeef859b2cab1494b9e26a95fc9eaf4f571fa00"],
61                      ["de7a7b30795cab310b632f708c6c2546847a5cbcc27ff48e75c1556c3f6f180c",
62                      "6218695558359d115e308b008d9aa368c38672732d2fc21c6317ad7d15918c05"],
63                      ["0ca70bbdea0e391b1e24e2540f33b48dd9dc554c61ebf23bb3691aab5094e40f",
64                      "dafecd436b2448504c0a3a1997b356c141f1d4b5977cc66e5f55592f13731501"]],
65                      "cc": "5059757cf06216215955aaa108e8dd40be157856749a9d883bcac611e395a409"
66              }]
67      }
68  }
69
```

Компоненты транзакции

- (строка 2) — print_tx указывает, где была найдена транзакция.
- (строка 3) — необработанные данные транзакции в нечитаемом формате.
- version (строка 5) — формат транзакции / era-версия; «2» соответствует протоколу RingCT.
- unlock_time (строка 6) — не позволяет тратить выходы транзакции до истечения определённого срока. Это либо высота блока, либо временная метка UNIX, если номер больше, чем начало времени UNIX.
- vin (строки 7-14) — список входов (в данном случае только один).
- amount (строка 9) — унаследованное поле суммы, используемое для транзакций первого типа.
- key_offset (строка 10) — позволяет верификаторам находить ключи и обязательства членов кольца в блокчейне и убедиться в том, что эти члены являются законными. Первый офсет является абсолютным в истории блокчейна, а каждый последующий офсет относительны предшествующего. Например, при реальных офсетах {7, 11, 15, 20} в блокчейн записываются {7, 4, 4, 5}. Верификаторы вычисляют последний офсет как {7+4+4+5=20} (см. Раздел 5.6.6).
- k_image (строка 12) — образ ключа $\widetilde{K}_j$ из Раздела 3.3, где $m = j = 1$ для 1 входа в данном случае.
- vout (строки 16-27) — список выходов (в данном случае их два).
- amount (строка 17) — унаследованное поле суммы, используемое для транзакций первого типа.

- key (строка 19) — одноразовый ключ места назначения для выхода *t*, как было описано в Разделе 5.2.
- extra (строки 28-32) — дополнительные данные, включающие в себя *публичный ключ транзакции*, то есть, совместно используемые секретные данные *rG* (см. Раздел 5.2), и ID платежа или зашифрованный ID платежа (см. Раздел 5.4). Это работает следующим образом: каждое число имеет размер один байт (оно может быть в диапазона от 0 до 255), и каждое может быть в поле «тег» или «размер». В теге указывается, какая информация будет следующей, а размер указывает, сколько байт эта информация занимает. Первое число всегда будет тегом. В данном случае «2» указывает на наличие «дополнительного нонса», что полезно для майнинг-пулов, а «33» будет указывать на то, что размер нонса составляет 33 байта. Проходит 33 числа, а затем следует новый тег, «1», который обозначает «публичный ключ транзакции». Размер публичных ключей транзакции всегда составляет 32 байта, поэтому нет никакой необходимости указывать размер. 32 числа идущие после являются концом этого дополнительного поля. (Примечание: в оригинальной спецификации первый байт указывал на размер поля. Monero не использует этого).
- rct signatures (строки 33-45) — первая часть данных подписи.
- type (строка 34) — тип подписи: RCTTypeFull является первым типом. RCTTypeSimple = 2, RCTTypeNull (майнинговая транзакция) = 0.
- txnFee (строка 35) — комиссия за транзакция, указанная простым текстом. В данном случае размер комиссии составляет 0,026 XMR.
- ecdhInfo (строки 36-42) — «информация по эллиптической кривой Диффи-Хелмана»: скрытая маска и сумма для каждого выхода $t \in \{1,\ldots,p\}$, где $p = 2$.
- mask (строка 37) — поле *mask* для $t = 1$, как было описано в Разделе 5.6.1.
- amount (строка 38) — поле *amount* для $t = 1$, как было описано в Разделе 5.6.1.
- outPk (строки 43-44) — обязательства по каждому выходу (см. Раздел 5.6.2).
- rctsig_prunable (строки 46-67) — вторая часть данных подписи.
- rangeSigs (строки 47-53) — доказательства диапазона для обязательств выхода $t \in 1,\ldots,p$.
- asig (строка 48) — список членов подписи Борромео (сокращённый вариант) для доказательства диапазона выхода $t = 1$ (включает все *c* и *r*), см. Раздел 5.6.6.
- Ci (строка 49) — список обязательств (сокращённый вариант) для доказательства диапазона выхода $t = 1$, как описано в Разделе 5.6.3. Так же, как говорилось в Разделе 5.6.6, только доказательства C_i требуют сохранения, так как значения $C_i - 2^i H$ легко могут быть вычислены верификаторами.
- MGs (строки 54-66) — остальные элементы подписи MLSAG.
- ss (строки 55-64) — компоненты $r_{i,j}$ подписи MLSAG

$$\sigma(m) = \left(c_1, r_{1,1}, \ldots, r_{1,m+1}, \ldots, r_{v+1,1}, \ldots, r_{v+1,m+1}\right)$$

- cc (строка 65) — компонент c_1 вышеупомянутой подписи MLSAG.

Приложение B

Структура транзакций RCTTypeSimple

В этом приложении мы приводим пример распечатанных данных реальной транзакции Monero типа RCTTypeSimple. У транзакции 4 входа и 2 выхода, и одна была добавлена в блокчейн с временной меткой 2017-12-21 11:36:20 UTC (как было указано майнером её блока).

```
print_tx 3ebf45fc5f8fd683037807384122817d5debfa762c7a7845cb7ccfe9ee20940b
Found in blockchain at height 1469563
020004[...]923b3d70d
{
        "version": 2,
        "unlock_time": 0,
        "vin": [{
                        "key":{
                        "amount": 0,
                         "key_offsets": [ 1567249, 1991110, 349235, 15551, 3620
                         ],
                         "k_image": "9661119b4b54529e1be14ef97fbdc0504d17a6c8dfedd55d2455b93a6336bb41"
         }
         },{
                         "key":{
                         "amount": 0,
                         "key_offsets": [ 2502375, 650851, 337433, 396459, 39529
                         ],
                        "k_image": "2102414d8edfa229f9ebf32ab90acd9cf23963a8c3b6ba0e181fc1d5782c046c"
        }
        },{
                        "key":{
                        "amount": 0,
                        "key_offsets": [ 1907097, 696508, 806254, 510195, 6709
                        ],
                        "k_image": "de14ec8958b311bd38a05aa3fb08fdd360001f1b9c060264eecdd8c08c9e83c4"
        }
        },{
                        "key":{
                        "amount": 0,
                        "key_offsets": [ 1150236, 1943388, 788506, 37175, 7462
                        ],
                        "k_image": "e470f77dd5a4149210cb61ee107e73caea1ef9f61d05384e3bd4372fdc85bf17"
                        }
        }
        ],
                        "vout": [{
                        "amount": 0,
                        "target":{
                        "key": "787cad1ebb181e1fc04b24d4d06c3d2882c38b262a7635de8ad487c536e40a12"
                        }
        },{
                                "amount": 0,
                                "target":{
                                "key": "faf4137928392b39ccf0a830c026157300995978769 7f9d4fb769c25781fb911"
                                }
                        }
        ],
        "extra": [ 1, 20, 56, 120, 111, 89, 89, 64, 10, 98, 96, 255, 202, 235, 203,
                        255, 2, 197, 176, 147, 61, 60, 41, 145, 207, 178, 212, 71, 37, 69, 19,
                        147, 205],
        "rct_signatures":{
        "type": 2,
        "txnFee": 558805800000,
        "pseudoOuts": [ "64dea29ac5560f93773240d58ca5768b879fd3c95e0b3b50a80ec36a6ff3a6da",
                                "a60e7a00e65ff2a6299b92b166a629e9b0d62f6df50e40535140716757efe4c0",
                                "4c67403adbc9dc0ca5a1a6abc846ab6d232dc3fa295099b3c7a9d005bac60eba",
                                "635b26d78117d77899859ecb61e10125c3956a5c113b932f33c92c561acddaa3"],
        "ecdhInfo": [{
                        "mask": "ccffac42a86bec7b36ce9957cbdfe481d419bc5353335d0c236c347aea758d0c",
                        "amount": "c0d6cf3e1db55dd459b73faf34d7339c3fa1b3d3356cfb2adc3faf798264b00e"
                        },{
                        "mask": "62cc846003d9c5425c6cf33b30754a4c044f5d9d02621460e45664b886673109",
                        "amount": "726dbacad62022bf0f5af05c72482b3f040d631d3f576b5e2615ea72f84c5f06"
                        }],
                        "outPk": [ "cb3b729b4fca6e66736666201633e3f905c367a2f3d18e31fe3d3c18d2be93fd",
                                        "1e8c86b7f211a99e1762bf62254efe65ea5c5328b62b0ea8d679b2e52800f633"]
```

```
68          },
69          "rctsig_prunable":{
70                  "rangeSigs": [{
71                          "asig": "9bb7cce09[...]61de7ce0a",
72                          "Ci": "50dfd2e8[...]b3a7c8fca1b1"
73                  },{
74                          "asig": "e3905fa5c[...]b5213444908",
75                          "Ci": "595c2cec5f2[...]72a628ab5c"
76                  }],
77                  "MGs": [{
78                          "ss": [ [ "3b2d26ea7628015fd8317e4e298ceda6b534ac894b83f7b6190a353cee6ec702",
79                                  "2d772ad7b7ff2ba8a1a66c8d69c0a0d49d72808eaf803c59f13c3d78b653440c"],
80                                  [ "c188891fb37d76305f0209222f52d22ede43018facfe91f949ecb8dcf709b30a",
81                                  "303896ca67ea7969544641d5bb94a436558bcf6522bb9bc77bd1abb5f2146c08"],
82                                  [ "e01c88b7308403a9dd023d9eacf1ade17ab0fa54250148431b5a33c98e636100",
83                                  "ba36e34e5245e89c7c21af845b949cf3e82188df639390f094e31c9ba773060c"],
84                                  [ "37175d72d2bef3f8bd9e65fb2861f7bce91e3b1e30278b2dcf26112831ac9405",
85                                  "8e77a5dd641a89e86dbe0708e8f59d0e2dc9fe4ddfd9b367c3a93522198a4706"],
86                                  [ "fbb4f94f9ce0f081421e63677a63d5914f0536a481d57b6e5fc5379c84dfcb05",
87                                  "1ec40aa3c8a94c6b1915b7796423b0d7d6011aa2d6af636aff309b832f193408"]],
88                          "cc": "a03119e4257cca37f89ac3e97f0598b712c79162c73932d58ab4ce08c4ad6709"
89                  },{
90                          "ss": [ [ "f7aedeec462d7588330c71589fde5f0f234a627a6e5ed72cff34825a04d41707",
91                                  "31d7a5ba4e782db5c0704ab751a2ef8c4732f3cf699bc8f9994e79a97cd3190e"],
92                                  [ "00e1d1ecdf31fdf7d57661f2234bfc859cfdc4dbfdfd0f5eec0576ef22592203",
93                                  "fdf08b803fa6de18bf0e0dc6855e877877bda0101eceb81e2223fe0175606300"],
94                                  [ "3397ba3f9e8db066e3c4911b896debefbc73efbac4988e6aff5731ff8db15405",
95                                  "43d2b03d5263de99f56c256e646be503edd63dd03d377a469379fbf487e8600e"],
96                                  [ "31afd1d5c3b07170ac127605fc35cbcc19cf963a35b2ff8f804e17e3b804000d",
97                                  "3a3bc124a10b0cf416656f8f682a427445140895440cca644c6aa38966399f0c"],
98                                  [ "f499f0e4922d5cba35e3cc033489b60ec7ea26ff19cc9dd29357670f4bf8790b",
99                                  "1a4732b31f0f1a7d3322be5c4baca098f0a032c192bf9f8a6b5fd83cbdd9d401"]],
100                         "cc": "095fcab7ebf64c2ffecdacf11b70f97f0e709de0a84b3a13abca627f9df2c901"
101                 },{
102                         "ss": [ [ "1808924b4154118c48f0b305562b6ffba86f38c64d4d8a087823f3383cddd006",
103                                 "4b18544be50aee8c4594b568d6be741c155a132cb83392d9b1a4cf35c3d5760c"],
104                                 [ "9a95eeeecf3c3a48c43873a372c263357ff5f258a7bf8ed29a767237b0b0f202",
105                                 "4f1ea4d9d4b56db780dd078a3e8219d0f54eaccc197901671002a206f063cb0e"],
106                                 [ "2c800017cab2b8388f58fed0d61a46570f64cacd8fabc4e84ddee735b3135f0a",
107                                 "458016f6fdf58b329fae0f929226ee2b8e410a14db8c6ede9b74fb718de71507"],
108                                 [ "ea0fcaf793602dce25c8b2c4d17163f4298933b3fb09874307d8cde9a63c2c0c",
109                                 "df76fbcd336c07f37e90e1a0d0db1ba49519ba4325062228bc9242af2c525703"],
110                                 [ "e3a7a0477eeb602a9a8203a6a496cca90c4769d57410246c4c8d665df34df900",
111                                 "44d206154f0ca85e12a92eefdbc3784e17e701a32ff93b550467679f67500c0d"]],
112                         "cc": "6f80de1c1d566776d2831f15c9a85fb1d8e8cecfd0d2753b318f0e84d89d3b08"
113                 },{
114                         "ss": [ [ "5b82b4644b57e3d623de7c72c6ebd52959815c12c80b479e4cbe5437cf67640c",
115                                 "b70e0b69c75faba6a1630429f9f497db351347c210467f69e1b1c5f1a72afe02"],
116                                 [ "f25a93a98f980cf489eb8f69369f4ec63eaea91fd677decab9b6ca0fe2feb606",
117                                 "124536c374cb6023a6aa6f22ae6e115a1ba12cb36c48f5f5ad43ce90f471da02"],
118                                 [ "151ddc82322456d7f31b8b4b2290098c3bf2428370c7ef325660b5463ff26404",
119                                 "2fb9d2979e16c2b1131686bb85068ec559f9c6c64581e609b451bb2cd9d5740d"],
120                                 [ "c03bed01d6ad60b3da5d2c88cf2e5023b51133c37e4917511715a11f09d8740d",
121                                 "432e01c2075ab6361af8636cc1c9254e12db98f5c323088792dfb42a1c894401"],
122                                 [ "52206d801214e70d20ee7ca53823c143aa06c3d1b22b118cc8a15c9f861f0102",
123                                 "563c134f56f7a290e0980877e93bc4b08651e53dade079b1e6c066b70fb81406"]],
124                         "cc": "4102cd245db3e0d7c0e2280cfdba38b9b7a7ad8715b8fe68c1170cf923b3d70d"
125                 }]
126         }
127 }
128
129
130
```


Компоненты транзакции

Далее следует разъяснение наиболее важных элементов этого типа транзакций. Мы упомянем только те компоненты, которые являются специфическими или отличаются от компонентов предыдущего типа транзакций RCTTypeFull.

- type (строка 53) — тип подписи, в этом случае имеющий значение 2 и соответствующий транзакциям типа RCTTypeSimple.
- pseudoOuts (строки 55-58) — обязательства по псевдо выходам $C_j^{'a}$, описанные в Разделе 5.7.1. Следует помнить о том, что сумма этих обязательств должна быть равной сумме обязательств по 2 выходам этой транзакции (плюс обязательству по комиссии за транзакцию *fH*).

Приложение C

Содержание блока

В этом приложении мы покажем структуру стандартного блока, а именно блока 1582196, если считать от генезис-блока. Блок содержит 5 транзакций, и был добавлен в блокчейн с временной меткой 2018-05-27 21:56:01 UTC (как было указано майнером блока).

```
1 print_block 1582196
2 timestamp: 1527458161
3 previous hash: 30bb9b475a08f2ea6fe07a1fd591ea209a7f633a400b2673b8835a975348b0eb
4 nonce: 2147489363
5 is orphan: 0
6 height: 1582196
7 depth: 2
8 hash: 50c8e5e51453c2ab85ef99d817e166540b40ef5fd2ed15ebc863091ca2a04594
9 difficulty: 51333809600
10 reward: 4634817937431
11 {
12          "major_version": 7,
13          "minor_version": 7,
14          "timestamp": 1527458161,
15          "prev_id": "30bb9b475a08f2ea6fe07a1fd591ea209a7f633a400b2673b8835a975348b0eb",
16          "nonce": 2147489363,
17          "miner_tx":{
18                      "version": 2,
19                      "unlock_time": 1582256,
20                      "vin": [{
21                                  "gen":{
22                                  "height": 1582196
23                                  }
24                      }
25          ],
26          "vout": [{
27                      "amount": 4634817937431,
28                      "target":{
29                                  "key": "39abd5f1c13dc6644d1c43db68691996bb3cd4a8619a37a227667cf2bf055401"
30                                  }
31                      }
32          ],
33          "extra": [ 1, 89, 148, 148, 232, 110, 49, 77, 175, 158, 102, 45, 72, 201, 193,
34          18, 142, 202, 224, 47, 73, 31, 207, 236, 251, 94, 179, 190, 71, 72, 251, 110,
35          134, 2, 8, 1, 242, 62, 180, 82, 253, 252, 0
36          ],
37          "rct_signatures":{
38                      "type": 0
39                      }
40          },
41          "tx_hashes": [ "e9620db41b6b4e9ee675f7bfdeb9b9774b92aca0c53219247b8f8c7aecf773ae",
42                                  "6d31593cd5680b849390f09d7ae70527653abb67d8e7fdca9e0154e5712591bf",
43                                  "329e9c0caf6c32b0b7bf60d1c03655156bf33c0b09b6a39889c2ff9a24e94a54",
44                                  "447c77a67adeb5dbf402183bc79201d6d6c2f65841ce95cf03621da5a6bffefc",
45                                  "90a698b0db89bbb0704a4ffa4179dc149f8f8d01269a85f46ccd7f0007167ee4"
46          ]
47 }
```

Компоненты блока

- (строки 2-10) — информация блока, собранная программным обеспечением. Откровенно говоря, она фактически не является частью блока.
- is orphan (строка 5) — указывает на то, что этот блок был заброшен. Обычно во время форка узлы сохраняют все ветви и отбрасывают их только в том случае, если появляется лидер по совокупной сложности. Таким образом, блоки становятся заброшенными.
- depth (строка 7) — в копии блокчейна глубиной любого заданного блока называют то, насколько далеко от самого последнего блока в блокчейне находится такой заданный блок.
- hash (строка 8) — это идентификатор (ID) блока.

- difficulty (строка 9) — значение сложности не сохраняется в блоке, так как пользователи могут вычислить сложность *всех* блоков на основе временных меток, и следуя правилам, описанным в Разделе 6.2.
- major_version (строка 12) — соответствует версии протокола, используемой для верификации этого блока.
- minor_version (строка 13) — изначально задумывалась как механизм голосования среди майнеров, а теперь просто является повторением major_version. Так как поле не занимает много места, возможно, что разработчики решили не тратить усилий на его удаление.
- timestamp (строка 14) — числовое выражение временной метки UTC этого блока, указываемой майнером блока.
- prev_id (строка 15) — ID предыдущего блока. В данном случае этим выражается сущность блокчейна Monero.
- nonce (строка 16) — нонс используется майнером блока для прохождения целевого значения сложности. Любой может повторно вычислить доказательство работы и проверить, является ли нонс действительным.
- miner_tx (строки 17-40) — транзакция майнера этого блока.
- version (строка 18) — формат транзакции / era-версия; «2» соответствует протоколу RingCT.
- unlock_time (строка 19) — выход транзакции майнера не может быть потрачен до тех пор, пока не будет вычислено ещё 60 блоков.
- vin (строки 20-25) — входы в майнинговую транзакцию. Здесь нет ни одного, так как майнинговая транзакция используется для генерирования вознаграждений за блоки и комиссий за транзакции.
- gen (строка 21) — сокращение от generate (генерировать).
- height (строка 22) — высота блока, для которой было сгенерировано вознаграждение за блок этой майнинговой транзакции. Вознаграждение за блок может быть сгенерировано только один раз для каждой высоты блока.
- vout (строки 26-32) — выходы майнинговой транзакции.
- amount (строка 27) — сумма, распределяемая майнинговой транзакцией, содержащая вознаграждение за блок и комиссии за транзакции в этом блоке. Записывается в атомных единицах.
- key (строка 29) — одноразовый адрес, указывающий принадлежность суммы, указанной в майнинговой транзакции.
- extra (строки 33-36) — дополнительная информация майнинговой транзакции, включая публичный ключ транзакции.
- type (строка 38) — тип транзакции (в этом случае это «0» (RCTTypeNull)), обозначающий майнинговую транзакцию.
- tx_hashes (строка 41) — все ID транзакций, входящих в этот блок (кроме ID майнинговой транзакции).

Приложение D

Генезис-блок

В этом приложении нами будет представлена структура генезис-блока Monero. Блок содержит 0 транзакций (с ним просто отправилось первое вознаграждение за блок пользователю thankful_for_today [63]). Основатель Monero не добавлял временной метки. Возможно, это было пережитком Bytecoin, монеты, в результате форка которой получился код Monero, и создатели которой очевидно пытались скрыть внушительный премайнинг [10].

Блок 1 был добавлен в блокчейн с временной меткой 2014-04-18 10:49:53 UTC (как было указано майнером блока), так что мы можем предположить, что генезис-блок был создан в тот же самый день. Это совпадает с датой запуска, заявленной thankful_for_today [63].

```
1 print_block 0
2 timestamp: 0
3 previous hash: 0000000000000000000000000000000000000000000000000000000000000000
4 nonce: 10000
5 is orphan: 0
6 height: 0
7 depth: 1580975
8 hash: 418015bb9ae982a1975da7d79277c2705727a56894ba0fb246adaabb1f4632e3
9 difficulty: 1
10 reward: 17592186044415
11 {
12          "major_version": 1,
13          "minor_version": 0,
14          "timestamp": 0,
15          "prev_id": "0000000000000000000000000000000000000000000000000000000000000000",
16          "nonce": 10000,
17          "miner_tx":{
18                      "version": 1,
19                      "unlock_time": 60,
20                      "vin": [{
21          "gen":{
22          "height": 0
23                                  }
24                      }
25          ],
26          "vout": [{
27                      "amount": 17592186044415,
28                      "target":{
29                      "key": "9b2e4c0281c0b02e7c53291a94d1d0cbff8883f8024f5142ee494ffbbd088071"
30                                  }
31                      }
32          ],
33          "extra": [ 1, 119, 103, 170, 252, 222, 155, 224, 13, 207, 208, 152, 113, 94, 188,
34          247, 244, 16, 218, 235, 197, 130, 253, 166, 157, 36, 162, 142, 157, 11, 200, 144,
35          209
36          ],
37          "signatures": [ ]
38          },
39          "tx_hashes": [ ]
40 }
```

Компоненты генезис-блока

Так как мы использовали одно и то же программное обеспечение для распечатки генезис-блока и блока, описанного в Приложении C, их структура практически одинакова. Но нам бы хотелось отметить некоторые важные отличия.

- difficulty (строка 9) — значение сложности генезис-блока указывается как 1, что означает, что может использоваться любой нонс.
- timestamp (строка 14) — у генезис-блока нет значимой временной метки.
- prev_id (строка 15) — нами используются пустые 32 байта в качестве ID предыдущего блока по определению.
- nonce (строка 16) — $n = 10000$ по определению.
- amount (строка 27) — точно соответствует нашим вычислениям вознаграждения за первый блок (17,592186044415 XMR) из Раздела 6.3.1.

- key (строка 29) — самые первые Monero были переданы их создателю thankful_for_today.
- extra (строка 33) — благодаря шифрованию, описанному в Приложении А, поле extra майнинговой транзакции генезис-блока содержит один публичный ключ транзакции.
- signatures (строка 37) — в генезис-блоке отсутствуют какие-либо подписи. В данном случае это просто артефакт функции print_block. То же самое относится и к tx_hashes в строке 38.