# От нуля к Monero: Второе издение

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

Опубликовано 4 апреля 2020 г. (v2.0.0)

koe[1], Курт М. Алонсо[2], Саранг Ноезер[3]
Перевод выполнен v1docq47[4]



[1] ukoe@protonmail.com
[2] kurt@oktav.se
[3] sarang.noether@protonmail.com
[4] chiptune@protonmail.ch

# Аннотация

Криптография. Может показаться, что эта во многом непонятная, эзотерическая и в чём-то элегантная тема доступна для понимания только математикам и учёным. На самом же деле многие виды криптографии достаточно просты, а их фундаментальные концепции доступны буквально каждому.

Общеизвестно, что криптография используется для защиты информации, будь то шифровка текстовых сообщений или обеспечение приватности цифрового взаимодействия. Ещё одним способом применения криптографии являются так называемые криптовалюты - цифровые деньги. Чтобы гарантировано исключить возможность дублирования или создания какой-либо части денег по собственному желанию. В своей основе криптовалюты обычно полагаются на «блокчейны», являющиеся публичными распределенными реестрами, которые содержат записи валютных транзакций, которые могут быть верифицированы третьими лицами [97].

Может показаться, что транзакции передаются и хранятся в простом текстовом формате, чтобы обеспечить возможность их публичной верификации. На самом же деле криптографические инструменты позволяют скрыть участников транзакции, а также передаваемые суммы, но при этом наблюдатели всё равно смогут верифицировать и согласовать их [136].

Мы стремимся научить любого, кто знаком с базовой алгеброй и простыми понятиями информатики, такими как «битовое представление» числа, не только тому, как работает Monero, но также и тому, насколько полезной и красивой может быть криптография.

Monero является криптовалютным блокчейном на базе одномерного распределённого ациклического графа (DAG) [97], где транзакции используют криптографию на эллиптических кривых, а именно кривой Ed25519 [38], входы транзакций подписываются при помощи многоуровневых связных подписей спонтанной анонимной группы (MLSAG) в стиле Шнорра [108], а суммы выходов (передаваемые получателям посредством ECDH [52]) скрыты при помощи обязательств Педерсена [88] и доказываются в допустимом диапазоне с использованием Bulletproofs [43].

# Содержание

**1 Введение** **1**

1.1 Цели . . . 2

1.2 Читателю . . . 4

1.3 Истоки криптовалюты Monero . . . 4

1.4 Структура документа . . . 5

1.4.1 Часть 1: «Основы» . . . 5

1.4.2 Часть 2: «Дополнения» . . . 5

1.4.3 Приложения . . . 6

1.5 Отказ от ответственности . . . 6

1.6 История «От нуля к Monero» . . . 6

1.7 Благодарность . . . 7

**I Основы** **8**

**2 Базовые концепции** **9**

2.1 Несколько слов о системе обозначений . . . 9

2.2 Модульная арифметика . . . 11

2.2.1 Сложение и умножение по модулю . . . 11
2.2.2 Возведение в степень по модулю . . . 13
2.2.3 Мультипликативная инверсия по модулю . . . 13
2.2.4 Уравнения с модулем . . . 14
2.3 Криптография на эллиптических кривых . . . 14
2.3.1 Что такое эллиптические кривые? . . . 14
2.3.2 Использование криптографии на эллиптических кривых для создания публичных ключей . . . 18
2.3.3 Протокол обмена ключами Диффи-Хеллмана на эллиптических кривых 18
2.3.4 Подписи Шнорра и преобразование Фиата-Шамира . . . 19
2.3.5 Подписание сообщений . . . 21
2.4 Кривая Ed25519 . . . 22
2.4.1 Двоичное представление . . . 23
2.4.2 Сжатие точек . . . 24
2.4.3 Алгоритм подписи EdDSA . . . 25
2.5 Двоичный оператор XOR . . . 26

**3 Продвинутые подписи Шнорра 28**
3.1 Доказательство знания дискретного логарифма на основе множества «базовых» ключей . . . 28
3.2 Множество приватных ключей в одном доказательстве . . . 30
3.3 Подписи спонтанной анонимной группы (SAG) . . . 31
3.4 Связываемые подписи спонтанной анонимной группы Бэка (bLSAG) . . . 33
3.5 Многоуровневые связываемые подписи спонтанной анонимной группы (MLSAG) 36
3.6 Компактные связываемые подписи спонтанной анонимной группы (CLSAG) . . 38

**4 Адреса Monero 42**
4.1 Ключи пользователя . . . 43
4.2 Одноразовые адреса . . . 43
4.2.1 Транзакции со множеством выходов . . . 45
4.3 Подадреса . . . 45
4.3.1 Отправка средств на подадрес . . . 46
4.4 Интегрированные адреса . . . 48
4.5 Адреса, использующие мультиподписи . . . 49

**5 Сокрытие суммы Monero** **50**

5.1 Обязательства . . . 50

5.2 Обязательства Педерсена . . . 51

5.3 Обязательства по сумме . . . 52

5.4 Введение в RingCT . . . 53

5.5 Доказательства диапазона . . . 54

**6 Кольцевые конфиденциальные транзакции (RingCT)** **56**

6.1 Типы транзакций . . . 57

6.2 Конфиденциальные транзакции `RCTTypeBulletproof2` . . . 57

6.2.1 Обязательства по сумме и комиссии за проведение транзакций . . . 57

6.2.2 Подпись . . . 58

6.2.3 Исключение возможности двойной траты . . . 60

6.2.4 Требования к занимаемому месту . . . 60

6.3 Краткое описание концепции . . . 62

6.3.1 Требования к занимаемому месту . . . 64

**7 Блокчейн Monero** **65**

7.1 Цифровая валюта . . . 66

7.1.1 Распределённая / совместно используемая версия событий . . . 66

7.1.2 Простой блокчейн . . . 67

7.2 Сложность . . . 68

7.2.1 Майнинг блока . . . 68

7.2.2 Скорость майнинга . . . 69

7.2.3 Консенсус: самая большая совокупная сложность . . . 70

7.2.4 Майнинг Monero . . . 70

7.3 Денежная масса . . . 72

7.3.1 Вознаграждение за майнинг блока . . . 73

7.3.2 Динамическое весовое значение блока . . . 74

7.3.3 Штраф, налагаемый на вознаграждение за блок . . . 77
7.3.4 Динамическая минимальная комиссия . . . 78
7.3.5 Последующая эмиссия . . . 81
7.3.6 Майнинговая транзакция: `RCTTypeNull` . . . 82
7.4 Структура блокчейна . . . 83
7.4.1 ID транзакции . . . 84
7.4.2 Дерево Меркла . . . 84
7.4.3 Блоки . . . 86

**II Расширения** **88**

**8 Доказательства знания в транзакциях Monero** **89**
8.1 Доказательства в транзакциях Monero . . . 89
8.1.1 Многоосновные доказательства в транзакциях Monero . . . 90
8.1.2 Доказательство создания входа транзакции (`SpendProofV1`) . . . 90
8.1.3 Создание доказательства по выходу транзакции (`OutProofV2`) . . . 92
8.1.4 Доказательство владения выходом (`InProofV2`) . . . 94
8.1.5 Доказательство того, что имеющийся выход не был потрачен в какой-либо транзакции (UnspentProof) . . . 95
8.1.6 Доказательство наличия минимального количества непотраченных средств по адресу (`ReserveProofV2`) . . . 97
8.2 Концепция аудита Monero . . . 99
8.2.1 Доказательство соответствия адреса и подадресов (SubaddressProof) . . . 100
8.2.2 Концепция аудита . . . 100

**9 Мультиподписи в Monero** **102**
9.1 Обмен данными с другими подписантами . . . 103
9.2 Агрегирование ключей для адресов . . . 104
9.2.1 Простейший подход . . . 104
9.2.2 Недостатки простейшего подхода . . . 104

9.2.3 Устойчивая агрегация ключей . . . 106
9.3 Пороговые подписи, подобные подписям Шнорра . . . 107
9.4 Кольцевые конфиденциальные подписи MLSTAG в Monero . . . 110
9.4.1 Транзакции `RCTTypeBulletproof2`, построенные по схеме multisig «N из N» . . . 110
9.4.2 Упрощённый вариант обмена данными . . . 113
9.5 Пересчёт образов ключей . . . 114
9.6 Меньшие пороговые значения . . . 116
9.6.1 Агрегирование ключей по схеме «1 из N» . . . 117
9.6.2 Агрегирование ключей по схеме «(N-1) из N» . . . 117
9.6.3 Агрегирование ключей по схеме «M из N» . . . 120
9.7 Семейства ключей . . . 122
9.7.1 Деревья семейств . . . 122
9.7.2 Вложение multisig-ключей . . . 123
9.7.3 Значение для Monero . . . 125

**10 Monero на рынке эскроу-услуг 126**
10.1 Важные особенности . . . 127
10.1.1 Процесс покупки . . . 128
10.2 Бесшовные мультиподписи Monero . . . 129
10.2.1 Основные принципы взаимодействия в рамках схемы multisig . . . 130
10.2.2 Пользовательский опыт взаимодействия с эскроу-сервисами . . . 132

**11 Объединённые транзакции Monero (TxTangle) 139**
11.1 Построение объединённых транзакций . . . 140
11.1.1 $n$-направленный канал передачи данных . . . 141
11.1.2 Количество раундов обмена сообщениями при построении объединённой транзакции . . . 141
11.1.3 Слабые стороны . . . 144
11.2 Организованная транзакция TxTangle . . . 145
11.2.1 Базовые принципы обмена данными с хостом через I2P и другие функции 145
11.2.2 Использование хоста в качестве сервиса . . . 147
11.3 Пользование услугами проверенного посредника . . . 148
11.3.1 Процедура с привлечением посредника . . . 148

**Список используемой литературы** 150

**Приложения** 159

**A Структура транзакций** `RCTTypeBulletproof2` 161

**B Содержание блока** 168

**C Генезис-блок** 172

# ГЛАВА 1

# Введение

В цифровой сфере сплошь и рядом создаются бесконечные копии информации с таким же бесконечным количеством изменений. Чтобы валюта смогла существовать в цифровом виде и получила широкое распространение, её пользователи должны верить в то, что её денежная масса строго ограничена. Получатель денег должен быть уверен, что он не получит поддельных монет или монет, которые уже были отправлены кому-то другому. Чтобы добиться этого без привлечения какой-либо третьей стороной, например, центрального руководящего органа, информация о денежной массе и полная история транзакций должны быть доступны для публичной проверки.

Мы можем использовать криптографические инструменты, чтобы регистрировать такую информацию в доступной базе данных, блокчейне, где она будет оставаться практически неизменной, где её будет невозможно подделать, а уровень её легитимности не сможет оспорить никакая из сторон.

Данные криптовалютных транзакций сохраняются в блокчейне, который служит публичным реестром[1] всех валютных операций. Большинство криптовалют сохраняет данные своих транзакций в виде простого текста, что упрощает процесс проверки (верификации) таких транзакций сообществом.

Очевидно, что открытый блокчейн не соответствует каким-либо принципам анонимности или

[1] В этом контексте реестр означает запись всех событий, связанных с созданием и обменом валюты. В частности, сколько денег было переведено в рамках каждого события и кому.

взаимозаменяемости[2], так как он буквально *придаёт огласке* полную историю всех транзакций, совершаемых его пользователями.

Для решения проблемы недостаточной приватности пользователи таких криптовалют, как Bitcoin, могут «маскировать» свои транзакции, используя временные промежуточные адреса [99]. Тем не менее, обладая соответствующими инструментами, можно проанализировать потоки и с большой степенью вероятности связать истинных отправителей с получателями [125, 41, 112, 48].

В отличие от таких валют Monero (Moe-neh-row) пытается решить проблему приватности путём сохранения в блокчейне только скрытых одноразовых адресов получателей средств. При этом подтверждение распределения средств в каждой транзакции осуществляется при помощи кольцевых подписей. Благодаря применению этих методов пока не было найдено никаких известных способов связать отправителей с получателями или отследить источник средств.[3]

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

Результатом является высокий уровень приватности и взаимозаменяемости криптовалюты.

## 1.1 Цели

Monero — это устоявшаяся криптовалюта, история разработки которой насчитывает уже более пяти лет [127, 13], и уровень распространённости которой постоянно растёт [54].[4] К сожалению, документации, подробно описывающей механизмы, используемые этой валютой, практически нет.[5,6] Хуже того, важные части теоретической концепции были опубликованы в

[2] «**Взаимозаменяемость** означает возможность взаимного замещения при использовании или выполнении контракта. ... Примеры: ... деньги и т. д.»[25] В случае с открытым блокчейном, таким как блокчейн Bitcoin, монеты, которыми владеет Элис, могут отличаться от монет, которыми владеет Боб из-за 'истории транзакций' таких монет. Если история транзакций монет Элис содержит информацию о транзакциях, предположительно связанных с незаконными действиями, её монеты могут быть 'помечены' [95], а следовательно, они будут менее ценными, чем монеты Боба (даже если это будет одна и та же сумма монет). Согласно заслуживающим доверия цифрам вновь созданные монеты Bitcoin торгуются с наценкой, если сравнивать их с монетами, имеющими историю, так как за ними просто не стоит никакой истории [118].

[3] В зависимости от поведения пользователей могут возникнуть ситуации, в которых транзакции можно будет в некоторой степени проанализировать. Примеры можно найти в этой статье: [61].

[4] С точки зрения рыночной капитализации, показатели Monero были стабильными относительно других криптовалют. По состоянию с 14 июня 2018 г. по 5 января 2020 г. Monero занимала 14-ю, позицию; см. https://coinmarketcap.com/.

[5] В одном из документов, опубликованных на https://monerodocs.org/, можно найти полезную информацию, в частности, связанную с интерфейсом командной строки (CLI). CLI-кошелёк поддерживает ввод команд посредством командной строки / консоли. Этот кошелёк имеет больше всего функций в сравнении с другими кошельками Monero, но они реализованы за счёт отсутствия удобного графического пользовательского интерфейса.

[6] Другой документ более общего характера под названием «Осваиваем Monero (Mastering Monero)» можно найти здесь: [57].

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.[7]

Кроме того, для тех, кто не имеет математического образования, изучение основ криптографии на эллиптических кривых, которую широко использует Monero, может оказаться совершенно бесцельным и разочаровывающим предприятием.[8]

Мы намерены исправить сложившуюся ситуацию, изложив фундаментальные концепции, необходимые для понимания криптографии на эллиптических кривых, рассмотрим алгоритмы и криптографические схемы, а также соберём воедино подробную информацию о внутренних механизмах работы Monero.

Чтобы нашим читателям было удобнее, мы постарались составить конструктивное, пошаговое описание криптовалюты Monero.

Во второй редакции этого отчета мы сосредоточили своё внимание на двенадцатой версии протокола Monero[9], что соответствует версии 0.15.x.x программного пакета Monero. Все описанные здесь механизмы, связанные с транзакциями и блокчейном, относятся именно к этим версиям.[10,11] Устаревшие схемы транзакций не анализировались нами, даже несмотря на то, что они могут частично поддерживаться из соображений обратной совместимости. То же относится и к устаревшим функциям блокчейна. Первая редакция [33] относится к седьмой версии протокола и версии 0.12.x.x программного пакета.

[7] Г-ном Сегиасом (Seguias) была написана превосходная серия статей под названием «Структурные элементы Monero» (Monero Building Blocks) [123], в которых подробно рассмотрены криптографические доказательства безопасности, используемые для обоснования схем подписей Monero. Как и в случае с первой редакцией «От нуля к Monero: Первое издаине» (Zero to Monero: First Edition) [33], серия статей Сегиаса сфокусирована на седьмой версии протокола.

[8] В рамках предыдущей попытки объяснить, как работает Monero [110], не была охвачена криптография эллиптических кривых, то есть работа была неполной, и теперь, спустя пять лет, её можно считать устаревшей.

[9] 'Протокол' представляет собой набор правил, согласно которым каждый блок тестируется перед тем, как он будет добавлен в блокчейн. Этот набор правил включает в себя 'протокол транзакций' (в настоящее время его вторую версию, RingCT) — это общие правила, определяющие принципы построения транзакций. Определённые правила, связанные с транзакциями, могут изменяться и изменяются без изменения версии протокола транзакций. Только масштабные изменения в структуре транзакций подразумевают изменение номера версии.

[10] Целостность и надёжность кодовой базы Monero обеспечиваются тем, что они были проанализированы достаточным количеством человек, выявивших все значительные ошибки. Мы надеемся, что читатели не воспримут наши объяснения как нечто, не требующее доказательств, и самостоятельно убедятся в том, что код работает именно так, как предполагается. Если будет иначе, мы надеемся, что вы ответственно раскроете эту информацию (`https://hackerone.com/monero`) , если это будет касаться каких-то серьёзных проблем, или создадите пул-реквест на GitHub (`https://github.com/monero-project/monero`), если речь будет идти о минимальных недочётах.

[11] В настоящий момент исследуется и анализируется несколько протоколов, заслуживающих рассмотрения в связи с транзакциями Monero следующего поколения. Это такие протоколы, как Triptych [106], RingCT3.0 [143], Omniring [84], и Lelantus [71].

## 1.2 Читателю

Мы ожидаем, что многие читатели, столкнувшиеся с этим отчетом, практически не будут иметь представления о том, что такое дискретная математика, алгебраические структуры, криптография[12] и блокчейны. Мы постарались достаточно подробно представить материал, чтобы непрофессионалы с любой точки зрения могли изучить Monero без проведения каких-либо внешних исследований.

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.[13]

## 1.3 Истоки криптовалюты Monero

Криптовалюта Monero, которая изначально называлась BitMonero, была создана в апреле 2014 в качестве производной криптовалюты на базе протокола доказательства концепции CryptoNote [127]. Monero на языке эсперанто означает «деньги», а форма множественного числа звучит как Moneroj (Moe-neh-rowje, похоже на Moneros, но с -ge на конце, как в английском слове orange).

CryptoNote является криптовалютным протоколом, разработанным самыми разными людьми. Первой значимой работой, в которой был описан этот протокол, стал документ, опубликованный в октябре 2013 [136]. Автор документа, который скрывался под псевдонимом Николас Ван Саберхаген (Nicolas van Saberhagen), предложил обеспечивать анонимность отправителей при помощи одноразовых адресов, а неотслеживаемость отправителей путём применения кольцевых подписей.

После этого аспекты приватности Monero усиливались за счёт реализации возможности сокрытия сумм, что описано Грэгом Максвеллом (Greg Maxwell) и другими авторами в работе [89], а также улучшения кольцевых подписей в соответствии с рекомендациями Шена Ноезера (Shen Noether) [108], и в результате эффективность подписей повысилась благодаря реализации Bulletproofs. [43].

[12] Исчерпывающий учебник по прикладной криптографии можно найти здесь: [42].

[13] Сноски, содержащиеся в некоторых главах, особенно в главах, связанных с протоколом, пересекаются с последующими главами и разделами. Они призваны сделать их понятнее при повторном прочтении, так как содержат некоторые подробности реализации, которые будут полезны тем, кто желает разобраться в том, как работает Monero.

## 1.4 Структура документа

Как уже было сказано ранее, нашей целью является создание полного и последовательного описания криптовалюты Monero. Структура «От нуля к Monero» соответствует этой задаче. Читатель последовательно знакомится со всеми подробностями внутренних принципов работы валюты.

### 1.4.1 Часть 1: «Основы»

Для обеспечения полноты описания мы решили представить все базовые криптографические элементы, необходимые для понимания сложностей, связанных с Monero, а также их математические основы. Во 2 главе нами рассматриваются важные аспекты криптографии на эллиптических кривых.

В 3 главе более широко рассмотрена схема подписи Шнорра, которой мы коснулись в предыдущей главе, а также кратко описаны алгоритмы построения кольцевых подписей, используемые для обеспечения конфиденциальности транзакций.

В 4 главе описано, как Monero использует адреса для контроля над обладанием средствами, а также рассматриваются различные виды адресов.

В 5 главе нами приводятся криптографические механизмы сокрытия сумм.

Наконец, после описания всех компонентов можно перейти и к схемам самих транзакций, используемых Monero, и 6 глава посвящена именно им.

Подробности, связанные с блокчейном Monero, раскрываются в 7 главе.

### 1.4.2 Часть 2: «Дополнения»

Криптовалюта — это больше, чем просто протокол, и в части «Дополнения» мы нами рассматривается ряд различных идей, многие из которых не были реализованы.[14]

Различная информация, связанная с транзакциями, может быть доказана перед наблюдателями, и соответствующие методы описаны в 8главе.

Хотя это и не важно с точки зрения работы Monero, мультиподписи очень полезны и позволяют множеству людей совместно отправлять и получать деньги. В 9 главе описан текущий подход Monero к применению мультиподписей, и кратко описаны возможные будущие разработки в этой области.

Чрезвычайно важно применять схему multisig при взаимодействии продавцов и покупателей на торговых онлайн площадках. В 10 главе показан пример нашего первоначального решения торговой эскроу-площадки с использованием схемы multisig, применяемой Monero.

[14] Следует отметить, что будущие версии протокола Monero, в частности, реализующие новые протоколы транзакций, могут сделать воплощение этих идей невозможным или непрактичным.

В этом отчёте в 11 главе мы впервые представляем TxTangle, децентрализованный протокол объединения транзакций множества пользователей в одну.

### 1.4.3 Приложения

В Приложении A рассматриваются примеры структур транзакций, взятых из блокчейна. В Приложении B разъясняется структура блоков (включая их заголовки и транзакции майнеров) в блокчейне Monero. Наконец, Приложение C завершает наш отчёт объяснением структуры генезис-блока Monero. Так мы попытались связать теорию, изложенную в предыдущих частях работы, с её реализацией на практике.

Мы используем примечания на полях, чтобы указать, где в исходном коде можно найти детали реализации Monero.[15] Как правило, существует путь к файлу, например src/ringct/rctOps.cpp, и функция, например `ecdhEncode()`. Примечание: '-' обозначает разделенный текст, например, crypto- note → cryptonote, и в большинстве случаев мы игнорируем операторы пространства имен (например `Blockchain::`).

Взгляните, разве это не полезно?

## 1.5 Отказ от ответственности

Все схемы подписей, варианты применения эллиптических кривых и подробности реализации Monero носят исключительно описательный характер. Читателям, исследующим серьёзные варианты практического применения (в отличие от исследователей-любителей), следует использовать первоисточники и технические спецификации (которые мы цитировали, где было возможно). Схемы подписей нуждаются в хорошо проверенных доказательствах безопасности, а детали реализации Monero можно найти в исходном коде Monero. В частности, как гласит поговорка, «не выдумывайте своей собственной криптографии». Криптографические примитивы, реализующие код, должны быть тщательно проанализированы экспертами и иметь долгую историю надёжной работы. Кроме того, оригинальные статьи, на которые ссылается этот документ, могут быть проанализированы в недостаточной мере и, вероятно, не проверены, поэтому читателям следует проявлять своё собственное суждение при их прочтении.

## 1.6 История «От нуля к Monero»

«От нуля к Monero» является как бы расширением магистерской диссертации Курта Алонсо под названием «Monero – технология обеспечения приватности блокчейна» [32], которая была опубликована в мае 2018 года. Первая редакция была опубликована в июне 2018 года [33].

[15] Наши сноски на полях точны с точки зрения версии 0.15.x.x программного пакета Monero, но постепенно могут утратить свою точность, так как кодовая база постоянно меняется. Тем не менее код сохраняется в соответствующем репозитории GitHub (`https://github.com/monero-project/monero`), что обеспечивает постоянный доступ к истории изменений.

Во второй редакции нами была улучшена часть, посвящённая применению кольцевых подписей (Глава 3), реорганизовано описание транзакций (добавлена Глава 4, посвящённая адресам Monero), обновлено описание метода передачи сумм выходов (раздел 5.3), ольцевые подписи Борромео заменены на Bulletproofs (раздел 5.5), исключены `RCTTypeFull` (Глава 6), обновлено и доработано описание динамических весовых значений блоков Monero и системы комиссий (Глава 7), рассмотрены доказательства, связанные с транзакциями (Глава 8), описаны мультиподписи Monero (Глава 9), предложены решения по реализации торговых эскроу-площадок (Глава 10), предложен новый протокол децентрализованных объединённых транзакций под названием TxTangle (Глава 11), с целью обеспечения соответствия текущей версии протокола (v12) и программному пакету Monero (v0.15.x.x) обновлены или добавлены различные незначительные подробности. Также из соображений удобочитаемости «вычищен» весь документ в целом.[16]

## 1.7 Благодарность

Написано автором ‘koe’.

Создание данного отчёта было бы невозможным без оригинальной магистерской диссертации Курта [32], и именно ему первому я выражаю благодарность. Исследователи из Исследовательской лаборатории Monero (MRL), Брэндон Гуддэл (Surae Noether), и псевдо-анонимный Саранг Ноезер (Sarang Noether) с которым мы сотрудничали при работе над разделом 5.5 и Главой 8), стали надёжным и исчерпывающим источником знаний при написании обеих редакций «От нуля к Monero». ‘moneromooo’, , самый плодовитый из ведущих разработчиков Проекта Monero, обладающий, пожалуй, самым глубоким знанием кодовой базы на этой планете, множество раз подталкивал меня в нужном направлении. И, конечно же, множество других замечательных контрибьюторов Monero потратило немало времени, отвечая на мои бесконечные вопросы. Наконец, хотелось бы поблагодарить тех многих людей, которые связывались с нами, чтобы внести коррективы — спасибо вам за ваши полезные и ободряющие комментарии!

[16] Исходный код LaTeX для обеих редакций «От нуля к Monero» можно найти здесь (первая редакция находится в ветке ‘ztm1’): https://github.com/UkoeHB/Monero-RCT-report.

# Часть I

# Основы

# ГЛАВА 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;
- символы верхнего 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. Более того, студенты-математики могут обратить внимание на то, что мы пренебрегли объяснениями, связанными с абстрактной алгеброй.

Тем не мене мы не считаем это упущением. Простые объекты, такие как целое число или последовательность, всегда могут быть представлены строкой бит. Такая вещь, как порядок байтов (endianness), встречается довольно редко и в большинстве случаев обозначается так, как это делается в наших алгоритмах.[1]

Точки эллиптической кривой 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.

src/crypto/keccak.c

Криптографическая хеш-функция (в дальнейшем именуемая просто «хеш-функцией» или «хешем») берёт некоторое сообщение $\mathfrak{m}$ произвольной длины и возвращает хеш $h$ (или «профиль сообщения») фиксированной длины, где каждый возможный выход равновероятно будет соответствовать определённому входу. Криптографические хеш-функции трудно обратить, так как они обладают интересным свойством, известным как *большой лавинный эффект (large avalanche effect)*, в силу которого очень схожие сообщения могут производить совершенно непохожие хеши, и поэтому довольно сложно найти два сообщения с одинаковым профилем.

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.

[1] В компьютерной памяти каждый байт сохраняется по собственному адресу (адрес сродни пронумерованному слоту, в котором может быть сохранён байт). Определённое «слово» или переменную можно найти по самому младшему адресу, имеющемуся в их байтах. Если переменная $x$ занимает 4 байта, сохранённые по адресам 10-13, то для нахождения $x$ используется адрес 10. То, как организованы байты $x$ в наборе адресов, зависит от *порядка байтов (endianness)*, даже несмотря на то, что каждый отдельно взятый байт всегда и везде сохраняется одинаково по своему адресу. Но с какого конца $x$ сохраняется по указанному адресу? $x$ может сохраняться *в прямом порядке*, с начала, или *в обратном порядке*, то есть с конца. Допустим $x$ = 0x12345678 (шестнадцатеричное представление, где 2 шестнадцатеричные цифры занимают 1 байт, например, 8 двоичных цифр, также известных, как биты), а массивом адресов является {10, 11, 12, 13}. При прямом порядке $x$ будет находиться по адресам {12, 34, 56, 78}, а при обратном порядке - {78, 56, 34, 12}. [78]

[2] Алгоритм хеширования Keccak лежит в основе стандарта Национального института стандартов и технологий США (NIST) *SHA-3* [27].

## 2.2 Модульная арифметика

Современная криптография по большей части начинается с модульной арифметики, которая, в свою очередь, начинается с операции по модулю (обозначаемой как 'mod'). Нас интересует только положительный модуль, который всегда возвращает положительное целое число.

Положительный модуль аналогичен «остатку» после деления двух чисел, например, $c$ является «остатком» $a/b$. Представим числовую ось. Чтобы вычислить $c = a \pmod b$ нам необходимо встать на точку $a$, а затем двигаться в направлении нуля, и каждый шаг должен быть равен step $= b$ , и так до тех пор, пока мы не достигнем целого числа $\geq 0$ и $< b$. То есть $c$. Например, 4 (modulo 3) $= 1$, $-5 \pmod 4 = 3$ и так далее.

Формально в данном случае модуль $c = a \pmod b$ определяется как $a = bx + c$, где $0 \leq c < b$, а $x$ является целым числом со знаком, которое отбрасывается ($b$ является положительным целым числом не равным нулю).

Следует отметить, что если $a \leq n$, $-a \pmod n$ будет тем же, что и $n - a$.

### 2.2.1 Сложение и умножение по модулю

В случае с компьютерной наукой важно избегать больших чисел при использовании модульной арифметики. Например, если у нас есть $29 + 87 \pmod{99}$ и мы не можем использовать переменные с тремя или большим количеством цифр (такие как $116 = 29 + 87$), то мы не можем вычислить $116 \pmod{99} = 17$ напрямую.

Чтобы выполнить действие $c = a+b \pmod n$, где оба значения $a$ и $b$ меньше модуля $n$, можно сделать следующее:

- вычислить $x = n - a$. Если $x > b$, значит $c = a + b$, в противном случае $c = b - x$.

Сложение по модулю можно использовать для получения умножения по модулю ($a * b \pmod n = c$) при помощи алгоритма под названием 'double-and-add'. Рассмотрим пример. Допустим, мы хотим выполнить действие $7 * 8 \pmod 9 = 2$. Это то же самое, что и

$$7 * 8 = 8 + 8 + 8 + 8 + 8 + 8 + 8 \pmod 9$$

Теперь разобьём это на группы по две:

$$(8 + 8) + (8 + 8) + (8 + 8) + 8$$

И ещё раз в группы по две:

$$[(8 + 8) + (8 + 8)] + (8 + 8) + 8$$

Общее количество операций сложения (+) точек сокращается с 6 до 4, поскольку найти сумму $(8 + 8)$ нам нужно лишь единожды.[3]

[3] Эффект применения алгоритма double-and-add становится очевидным при работе с большими числами. Например, при умножении $2^{15} * 2^{30}$ прямое сложение потребует примерно $2^{15}$ + операций сложения, в то время как при использовании алгоритма количество операций сокращается всего до 15!

Алгоритм реализуется путём преобразования первого числа («множителя» $a$) в двоичную форму (например, $7 \rightarrow [0111]$) с последующим проходом через двоичный массив и выполнением операций удваивания и сложения.

Создадим массив $A = [0111]$ и пронумеруем его как 3,2,1,0.[4] A[0] = 1 является первым элементом A и наименьшим значащим битом. Изначально задаём полученную переменную как $r = 0$ и переменную сумму как $s = 8$ (в более общем смысле мы начинаем с $s = b$). Далее следуем алгоритму:

1. Выполняем итерацию: $i = (0, ..., A_{size} - 1)$
   (a) Если A[i] == 1, значит, $r = r + s \pmod{n}$.
   (b) Вычисляем $s = s + s \pmod{n}$.
2. Используем полученное $r$: $c = r$.

В нашем примере $7 * 8 \pmod 9$ появляется эта последовательность:

1. $i = 0$
   (a) A[0] = 1, таким образом, $r = 0 + 8 \pmod 9 = 8$
   (b) $s = 8 + 8 \pmod 9 = 7$
2. $i = 1$
   (a) A[1] = 1, таким образом, $r = 8 + 7 \pmod 9 = 6$
   (b) $s = 7 + 7 \pmod 9 = 5$
3. $i = 2$
   (a) A[2] = 1, таким образом, $r = 6 + 5 \pmod 9 = 2$
   (b) $s = 5 + 5 \pmod 9 = 1$
4. $i = 3$
   (a) A[3] = 0, таким образом, $r$ остаётся тем же
   (b) $s = 1 + 1 \pmod 9 = 2$
5. Получаем $r = 2$

[4] Эта операция известна как нумерация ‘LSB 0’, так как наименьший значащий бит имеет значение 0. Мы используем ‘LSB 0’ во всех остальных случаях, рассмотренных в данной главе, для ясности, а не для точности обозначения.

### 2.2.2 Возведение в степень по модулю

Очевидно, $8^7 \pmod 9 = 8*8*8*8*8*8*8 \pmod 9$. Подобно тому как это делается в случае с алгоритмом double-and-add, мы можем произвести возведение в квадрат и умножение (square-and-double). Для $a^e \pmod n$:

1. Определяем $e_{scalar} \to e_{binary}$; $A = [e_{binary}]$; $r = 1$; $m = a$
2. Выполняем итерацию: $i = (0, ..., A_{size} - 1)$
   (a) Если A[i] == 1, значит, $r = r * m \pmod n$.
   (b) Вычисляем $m = m * m \pmod n$.
3. Используем полученное $r$ в качестве результата.

### 2.2.3 Мультипликативная инверсия по модулю

Иногда нам требуется $1/a \pmod n$ или, другими словами, $a^{-1} \pmod n$. Инверсия чего-либо умноженного сколько-то раз на себя по определению равна 1 (единице). Возьмём $0.25 = 1/4$, а затем $0.25 * 4 = 1$.

В модельной арифметике, в случае $c = a^{-1} \pmod n$, $ac \equiv 1 \pmod n$ для $0 \leq c < n$ и для $a$ и $n$ будут взаимно-простыми числами.[5] «Взаимно простые» означает, что они не будут иметь никаких общих делителей, кроме 1 (фракция $a/n$ не может быть сокращена/упрощена).

Мы можем использовать алгоритм возведения в квадрат и умножения для вычисления мультипликативной инверсии по модулю, если $n$ является простым числом в соответствии с *малой теоремой Ферма*:[6]

$$\begin{aligned} a^{n-1} &\equiv 1 \pmod n \\ a * a^{n-2} &\equiv 1 \pmod n \\ c \equiv a^{n-2} &\equiv a^{-1} \pmod n \end{aligned}$$

В более общем смысле (и более кратко) так называемый «расширенный алгоритм Евклида» [7] также позволяет найти инверсии по модулю.

[5] В уравнении $a \equiv b \pmod n$, $a$ а является *конгруентным* для $b \pmod n$, что просто означает, что $a \pmod n = b \pmod n$.

[6] Мультипликативная инверсия по модулю подчиняется следующему правилу:
*Если $ac \equiv b \pmod n$, где $a$ и $n$ являются взаимно простыми числами, решение такой линейной конгруэнции задаётся выражением $c = a^{-1}b \pmod n$.*[10]
Это означает, что мы можем выполнить $c = a^{-1}b \pmod n \to ca \equiv b \pmod n \to a \equiv c^{-1}b \pmod n$.

### 2.2.4 Уравнения с модулем

Предположим, у нас есть уравнение $c = 3 * 4 * 5 \pmod 9$. Оно вычисляется напрямую. У нас имеется некоторая операция $\circ$ (например, $\circ = *$) между двумя выражениями $A$ и $B$:

$$(A \circ B) \pmod n = [A \pmod n] \circ [B \pmod n] \pmod n$$

В нашем примере мы задаём $A = 3 * 4$, $B = 5$ и $n = 9$:

$$\begin{aligned}(3 * 4 * 5) \pmod 9 &= [3 * 4 \pmod 9] * [5 \pmod 9] \pmod 9 \\ &= [3] * [5] \pmod 9 \\ c &= 6\end{aligned}$$

Теперь можно выполнить вычитание по модулю:

$$\begin{aligned}A - B \pmod n &\to A + (-B) \pmod n \\ &\to [A \pmod n] + [-B \pmod n] \pmod n\end{aligned}$$

Тот же самый принцип применим к выражениям, подобным $x = (a - b * c * d)^{-1}(e * f + g^h) \pmod n$.[7]

## 2.3 Криптография на эллиптических кривых

### 2.3.1 Что такое эллиптические кривые?

`fe`: элемент поля

Конечное поле $\mathbb{F}_q$, где $q$ является простым числом более 3, это поле, сформированное последовательностью $\{0, 1, 2, ..., q - 1\}$. Арифметические действия $(+, \cdot)$ и унарная операция $(-)$ являются вычисленным $\pmod q$.

«Вычисленный $\pmod q$» означает $\pmod q$, который производится для каждого вида арифметического действия между двумя элементами поля или для отрицания отдельно взятого элемента поля. Например, при заданном простом поле $\mathbb{F}_p$, где $p = 29$, $17 + 20 = 8$, так как $37 \pmod{29} = 8$. Точно так же $-13 = -13 \pmod{29} = 16$.

[7] Модуль больших чисел не может использовать уравнения с модулем. Получается $254 \pmod{13} \equiv 2 * 10 * 10 + 5 * 10 + 4 \equiv (((2) * 10 + 5) * 10 + 4) \pmod{13}$. Алгоритм вычисления $a \pmod n$, если $a > n$, будет следующим:

1. Определяем $A \to [a_{decimal}]$; $r = 0$
2. Для $i = A_{size} - 1, ..., 0$
   (a) $r = (r * 10 + A[i]) \pmod n$
3. Используем полученное $r$ в качестве результата.

Обычно эллиптические кривые определяют как набор точек $(x, y)$, соответствующих уравнению *Вейерштрасса* для заданной пары $(a, b)$ [69]:[8]

$$y^2 = x^3 + ax + b \quad \text{где} \quad a, b, x, y \in \mathbb{F}_q$$

Криптовалюта Monero использует специальную кривую, принадлежащую к категории так называемых *скрученных кривых Эдвардса (Twisted Edwards curves)* [38], которые обычно представлены следующим выражением (для заданной пары $(a, d)$):

$$ax^2 + y^2 = 1 + dx^2y^2 \quad \text{где} \quad a, d, x, y \in \mathbb{F}_q$$

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 Бернштейна [40] *и др.*

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). Производим сложение по точкам, определяя $P_1 + P_2 = (x_1, y_1) + (x_2, y_2)$ как $P_3 = (x_3, y_3)$, где[9]

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \pmod q$$
$$y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \pmod q$$

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 по оси $y$ $(x, y) \to (-x, y)$ [38] и добавить точку. При каждом появлении «отрицательного» элемента $-x$ поля $\mathbb{F}_q$ в этом отчёте фактически это будет $-x \pmod q$.

Всякий раз при сложении точек кривой $P_3$ является точкой на «оригинальной» эллиптической кривой, или, другими словами, все $x_3, y_3 \in \mathbb{F}_q$ соответствуют условиям уравнения EC.

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 $(0, P, 2P, 3P, 4P)$, 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 $(0, 1)$.[10]

[8] Примечание: выражение $a \in \mathbb{F}$ означает, что $a$ является некоторым элементом в поле $\mathbb{F}$.

[9] Как правило, точки на эллиптической кривой преобразуются в проективные координаты до выполнения операций на кривой, таких как сложение точек, чтобы избежать выполнения инверсий поля с целью повышения эффективности. [142]

src/crypto/crypto_ops_builder/ref10CommentedCombined/ge.h

[10] В результате использования описанной операции добавления эллиптические кривые приобретают структуру *абелевой группы*, так как их бесконечно удалённые точки являются единичными элементами. Точное определение этого понятия можно прочитать по ссылке https://brilliant.org/wiki/abelian-group/.

Это очень удобно: $5P + P = P$. Это указывает на *цикличность*[11] подгруппы. Все точки $P$ кривой EC формируют циклические подгруппы. Если точка $P$ формирует подгруппу, то все входящие в эту подгруппу точки (за исключением бесконечно удалённой точки) также формируют такую подгруппу. В нашем примере взяты кратные значения точки $2P$:

$$2P, 4P, 6P, 8P, 10P \rightarrow 2P, 4P, 1P, 3P, 0$$

Другой пример: подгруппа порядка 6 $(0, P, 2P, 3P, 4P, 5P)$. Кратными точки $2P$ будут:

$$2P, 4P, 6P, 8P, 10P, 12P \rightarrow 2P, 4P, 0, 2P, 4P, 0$$

В данном случае $2P$ имеет порядок 3. Так как 6 не является простым числом, не все входящие сюда точки смогу воссоздать оригинальную 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*). Можно представить себе набор из всех точек EC $\{0, P_1, ..., P_{N-1}\}$. Если $N$ не является простым числом, некоторые точки создадут подгруппы с порядками, равными делителям $N$.

Чтобы определить порядок $u$ подгруппы точки $P$, необходимо:

1. Найти $N$ (например, используя *алгоритм Шуфа*).
2. Найти все делители $N$.
3. Для каждого делителя $n$ у $N$ найти $nP$.
4. Самый малый делитель $n$, при котором $nP = 0$ будет порядком $u$ подгруппы.

ge: групповой элемент

У кривых EC, выбранных для криптографии, обычно $N = hl$, где $l$ является достаточно большим простым числом (как 160 бит), а $h$ является так называемым *кофактором*, который будет иметь малое значение, равное 1 или 2.[12] Одна точка из подгруппы с размером $l$ выбирается в качестве генератора и обозначается как $G$. Для каждой другой точки $P$ в подгруппе существует целое число $0 < n \leq l$, соответствующее $P = nG$.

Давайте рассмотрим это подробнее для понимания. Допустим, у нас есть точка $P'$ порядка $N$, где $N = hl$. Любая другая точка $P_i$ может быть найдена при помощи некоторого целого числа $n_i$ так, чтобы $P_i = n_i P'$. 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

[11] Цикличность подгруппы означает, что для подгруппы точки $P$ порядка $u$ и любого целого числаг $n$, $nP = [n \pmod u]P$. Можно представить, что мы находимся в какой-то точке земного шара, находящейся на некотором расстоянии от «нулевой» метки, и каждый шаг, который мы делаем, смещает нас на это расстояние. В какой-то момент мы вернёмся в начальную точку, даже несмотря на то, что до того момента, как мы точно попадём в исходную точку, нам придётся несколько раз обернуться вокруг земного шара. Количество шагов, необходимых для того, чтобы попасть в ту же самую точку, будет «порядком» «группы шагов», и все наши следы будут уникальными точками в этой группе. Рекомендуем использовать эту концепцию применительно к остальным идеям, изложенным в данной работе.

[12] EC с небольшими кофакторами обеспечивает возможность относительно быстрого добавления точек и т. д. [38].

если $l(n_1P') \equiv l(n_2P') \equiv NP' = 0$, то $n_1$ и $n_2$ будут кратными $h$. Так как $N = hl$, значит $l$ будет кратным $h$, что подразумевает возможность существования только одной подгруппы с размером $l$.

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 будет только $h$ таких вариантов $n' \pmod N$ (включая бесконечно удалённую точку для $n' = hl$), поскольку, если $n' = hl$ происходит обратное возвращение по циклу к 0: $hlP' = 0$. Таким образом, если $hP$ равно 0, будет только $h$ точек $P$ на EC.

Подобный аргумент можно применить и к любой другой подгруппе размера $u$. Любые две точки $P_1$ и $P_2$ порядка $u$ находятся в одной и той же подгруппе, состоящей из кратных $(N/u)P'$.

Учитывая это новое объяснение, очевидно, мы можем использовать следующий алгоритм для вычисления точек (не являющихся бесконечно удалёнными) в подгруппе порядка $l$:

1. Найти $N$ эллиптической кривой EC, выбрать подгруппу порядка $l$, вычислить $h = N/l$.
2. Выбрать произвольную точку $P'$ на кривой EC.
3. Вычислить $P = hP'$.
4. Если $P = 0$, вернуться к шагу 2, или же $P$ формирует подгруппу порядка $l$.

Вычисление скалярного произведения любого целого числа $n$ и любой точки $P$, $nP$, не представляет сложности, в то время как нахождение такого $n$, чтобы $P_1 = nP_2$, является сложным с точки зрения вычисления. По аналогии с модульной арифметикой эту задачу часто называют *задачей дискретного логарифмирования (discrete logarithm problem, DLP)*. Другими словами, скалярное умножение может рассматриваться в качестве *необратимой функции*, что позволяет использовать эллиптические кривые в криптографии.[13]

sc: скаляр

Скалярное произведение $nP$ эквивалентно $(((P+P)+(P+P))\ldots)$. Несмотря на то, что этот подход не всегда является наиболее эффективным, мы можем воспользоваться алгоритмом double-and-add, как это было сделано в разделе 2.2.1. Чтобы получить сумму $R = nP$, , следует помнить о том, что мы используем операцию сложения точек, о которой говорится в разделе 2.3.1.

1. Определяем $n_{scalar} \to n_{binary}$; $A = [n_{binary}]$; $R = 0$, бесконечно удалённую точку; $S = P$
2. Выполняем итерацию: $i = (0, ..., A_{size} - 1)$
   (a) Если A[i] == 1, значит, R += S.

[13] Не существует известного алгоритма, позволяющего эффективно (при помощи доступной технологии) найти решения для $n$ при $P_1 = nP_2$, а это означает, что для разбора всего одного скалярного произведения уйдут многие годы.

   (b) Вычисляем S += S.

3. Используем полученное R в качестве результата.

Следует отметить, что скалярные величины EC точек в подгруппе размера $l$ (которая будет использоваться в дальнейшем) являются элементами конечного поля $\mathbb{F}_l$. Это означает, что арифметика между скалярами равна mod $l$.

### 2.3.2 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 $0 < k < l$, и мы назовём его *приватным ключом.*[14] Необходимо вычислить соответствующий *публичный ключ* $K$ (точку на кривой EC), используя скалярное произведение $kG = K$.

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

### 2.3.3 Протокол обмена ключами Диффи-Хеллмана на эллиптических кривых

Базовый обмен секретными ключами *Диффи-Хеллмана* [52] между *Элис* и *Бобом* мог бы происходить 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.

   Например, если у Элис имеется сообщение $m$, которое она хочет отправить Бобу, она может хешировать общий секрет $h = \mathcal{H}(S)$, вычислить $x = m + h$, и отправить $x$ Бобу. Боб вычисляет $h' = \mathcal{H}(S)$, вычисляет $m = x - h'$ и узнаёт $m$.

[14] Иногда приватный ключ называют *секретным ключом.* Поэтому мы используем сокращения: pk = публичный ключ (public key), sk = секретный ключ (secret key).

Сторонний наблюдатель не сможет вычислить общий секрет благодаря задаче Диффи-Хеллмана DHP, согласно которой найти $S$ на основе $K_A$ и $K_B$ очень сложно. Также DLP не позволяет им найти $k_A$ или $k_B$.[15]

### 2.3.4 Подписи Шнорра и преобразование Фиата-Шамира

В 1989 году Клаусом Питером Шнорром был опубликован теперь уже ставший известным протокол интерактивной аутентификации [121], затем обобщённый Маурером в 2009 [87], который позволял кому-либо доказывать знание приватного ключа $k$ для определённого публичного ключа $K$, не раскрывая никакой информации о нём [89]. Выглядит это примерно так:

1. Доказывающая сторона генерирует случайное целое число $\alpha \in_R \mathbb{Z}_l$[16], вычисляет $\alpha G$ и отправляет $\alpha G$ верификатору.
2. Верификатор генерирует случайный *запрос* $c \in_R \mathbb{Z}_l$ и отправляет $c$ проверяющей стороне.
3. Доказывающая сторона вычисляет *ответ* $r = \alpha + c * k$ и отправляет $r$ верификатору.
4. Верификатор вычисляет $R = rG$ и $R' = \alpha G + c * K$ и проверяет $R \stackrel{?}{=} R'$.

Верификатор может вычислить $R' = \alpha G + c * K$ до доказывающей стороны, передав $c$ и как бы говоря: «Я запрашиваю ответа в форме дискретного логарифма $R'$». На такой запрос доказывающая сторона может ответить, только зная $k$ (за исключением ничтожной вероятности).

Если значение $\alpha$ было выбрано случайным образом, то $r$ распределяется случайным образом [122], а $k$ является теоретико-информационно безопасным в пределах $r$ (его по-прежнему можно найти, решив задачу DLP для $K$ или $\alpha G$).[17] Тем не менее, если доказывающая сторона повторно использует $\alpha$, чтобы доказать знание $k$, любой, кому известны оба запроса в $r = \alpha + c * k$ и $r' = \alpha + c' * k$, сможет вычислить $k$ (два уравнения, два неизвестных).[18]

$$k = \frac{r - r'}{c - c'}$$

[15] Предполагается, что задача DHP должна иметь, по крайней мере, ту же сложность, что и DLP, хотя это и не было доказано. [31]

[16] Примечание: $R$ в $\alpha \in_R \mathbb{Z}_l$ означает, что $\alpha$ случайным образом выбирается из $\{1, 2, 3, ..., l-1\}$. Другими словами, $\mathbb{Z}_l$ является всеми числами (mod $l$). Мы исключаем '$l$' поскольку в данном случае бесконечно удалённая точка не требуется. src/crypto/crypto.cpp `random32_unbiased()`

[17] Криптосистемой, обеспечивающей теоретико-информационную безопасность, является та система, которая не может быть взломана злоумышленником, обладающим даже безграничным вычислительным ресурсом, просто потому, что у него не будет достаточного количества информации.

[18] Если доказывающей стороной является компьютер, можете представить кого-то «клонирующего»/копирующего компьютер после того, как он сгенерирует $\alpha$, а затем передающего каждую копию с отдельным запросом.

Если доказывающей стороне с самого начала было известно $c$ (например, если верификатор секретно передал его доказывающей стороне), гона должна сгенерировать случайный ответ $r$ и вычислить $\alpha G = rG - cK$. Когда она позже отправит $r$ верификатору, она докажет знание $k$, даже не зная его. Кто-то, просматривающий запись событий между доказывающей стороной и верификатором, также ничего не узнает. Схему нельзя *верифицировать публично.* [89]

В роли запросчика верификатор после получения $\alpha G$ выдаёт случайное число, что делает его эквивалентным *случайной функции.* Случайные функции, такие как хеш-функция, называются случайными оракулами, поскольку их вычисление подобно запросу у кого-то случайного числа [89].[19]

Использование хеш-функции вместо верификатора для формирования вызовов известно как *преобразование Фиата-Шамира* [60], поскольку таким образом интерактивное доказательство превращается в неинтерактивное и публично верифицируемое [89].[20,21]

**Неинтерактивное доказательство**

1. Генерирование случайного числа $\alpha \in_R \mathbb{Z}_l$ и вычисление $\alpha G$.
2. Вычисление запроса при помощи криптографически защищённой хеш-функции, $c = \mathcal{H}([\alpha G])$.
3. Определение ответа $r = \alpha + c * k$.
4. Публикация пары доказательства $(\alpha G, r)$.

**Верификация**

1. Вычисление запроса: $c' = \mathcal{H}([\alpha G])$.
2. Вычисление $R = rG$ и $R' = \alpha G + c' * K$.
3. Если $R = R'$, значит, доказывающей стороне известен $k$ (за исключением ничтожной вероятности).

[19] В более общем смысле «в криптографии… оракул является любой системой, которая может дать дополнительную информацию по системе, которая в противном случае не была бы известна».[3]

[20] Выход криптографической хеш-функции $\mathcal{H}$ единообразно распределяется среди ряда возможных выходов. Другими словами, для некоторого входа $A$, $\mathcal{H}(A) \in_R^D \mathbb{S}_H$, где $\mathbb{S}_H$ является рядом возможных выходов $\mathcal{H}$. Мы используем $\in_R^D$, чтобы указать на то, что функция является детерминированно случайной. $\mathcal{H}(A)$ каждый раз даёт тот же результат, но её выход эквивалентен случайному числу.

[21] Следует отметить, что неинтерактивные доказательства (и подписи), подобные доказательствам Шнорра, либо используют фиксированный генератор $G$, либо включают генератор в хеш запроса. Такое включение известно как прибавление префикса ключа, о чём будет говориться чуть позже (разделы 3.4 и 9.2.3).

**Почему это работает**

$$\begin{aligned} rG &= (\alpha + c * k)G \\ &= (\alpha G) + (c * kG) \\ &= \alpha G + c * K \\ R &= R' \end{aligned}$$

Важной составляющей любой схемы доказательства/подписи являются ресурсы, необходимые для их верификации. Они включают в себя место, необходимое для сохранения доказательств, и время, необходимое для верификации. В случае с данной схемой мы сохраняем точку EC и одно целое число, и нам необходимо знать публичный ключ — другую точку EC. Поскольку вычисление хеш-функций происходит относительно быстро, следует помнить о том, что время верификации по большей части является функцией операций на эллиптической кривой. src/ringct/rctOps.cpp

### 2.3.5 Подписание сообщений

Обычно криптографическая подпись строится на криптографическом хеше сообщения, а не на самом сообщении, что облегчает подписание сообщений разного размера. Тем не менее в этом отчёте нами будет свободно использоваться термин «сообщение» и символ $\mathfrak{m}$ для определения сообщения и/или его значения хеша, если не будет указано иное.

Подписание сообщений является основой интернет-безопасности и даёт получателю уверенность в том, что его содержание является именно тем, что было определено отправителем. Одной из наиболее широко используемых подписей является ECDSA (см. [72], ANSI X9.62 и [69].

Схема подписи, о которой мы будем говорить в данном случае, является альтернативной формулировкой преобразованного доказательства Шнорра, о котором было сказано выше. Если рассматривать подписи под таким углом, будет понятнее то, что будет говориться о кольцевых подписях в следующей главе.

**Подпись**

Предположим, у Элис есть пара, состоящая из приватного/публичного ключей $(k_A, K_A)$. Чтобы уникально подписать случайное сообщение $\mathfrak{m}$, она могла бы сделать следующее:

1. Генерирование случайного числа $\alpha \in_R \mathbb{Z}_l$ и вычисление $\alpha G$.
2. Вычисление запроса при помощи криптографически защищённой хеш-функции, $c = \mathcal{H}(\mathfrak{m}, [\alpha G])$.
3. Определение такого $r$, чтобы $\alpha = r + c * k_A$. Другими словами, $r = \alpha - c * k_A$.
4. Публикация подписи $(c, r)$.

#### Верификация

Любая третья сторона, которой известны параметры кривой EC (указывающие на то, какая эллиптическая кривая использовалась), подпись $(c, r)$ и метод подписания, $\mathfrak{m}$ и хеш-функция, а также $K_A$ , может верифицировать подпись:

1. Вычисление запроса: $c' = \mathcal{H}(\mathfrak{m}, [rG + c * K_A])$.
2. Если $c = c'$, подпись проходит верификацию.

В случае с данной схемой подписи мы сохраняем две скалярные величины, и нам нужен один публичный ключ EC.

#### Почему это работает

Причиной является тот факт, что

$$\begin{aligned} rG &= (\alpha - c * k_A)G \\ &= \alpha G - c * K_A \\ \alpha G &= rG + c * K_A \\ \mathcal{H}_n(\mathfrak{m}, [\alpha G]) &= \mathcal{H}_n(\mathfrak{m}, [rG + c * K_A]) \\ c &= c' \end{aligned}$$

Следовательно, обладатель $k_A$ (Элис) создал $(c, r)$ на основе $\mathfrak{m}$: она подписала сообщение. Вероятность того, что кто-то ещё, злоумышленник, пытающийся создать подделку, не зная $k_A$, сможет сделать $(c, r)$, ничтожна, поэтому верификатор может быть уверен в том, что сообщение не было сфальсифицировано.

## 2.4 Кривая Ed25519

Для выполнения криптографических операций Monero использует определённую скрученную эллиптическую кривую Эдвардса, *Ed25519*, *бирациональный эквивалент*[22] кривой Монтгомери *Curve25519*.

Как кривая Curve25519, так и кривая Ed25519 были описаны в работах Бернштейна *и др.* [38, 39, 40].[23]

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

[23] Доктором Бернштейном была также разработана схема шифрования, известная как ChaCha [37, 100], которая использовалась первым вариантом реализации Monero для шифрования определённой чувствительной информации, связанной с «кошельками пользователей». src/wallet/ringdb.cpp

Кривая определяется через простое поле $\mathbb{F}_{2^{255}-19}$ (то есть $q = 2^{255}-19$) при помощи следующего уравнения:

src/crypto/crypto_ops_builder/ref10CommentedCombined/ge.h

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

Использование этой кривой решает ряд вопросов, поднятых криптографическим сообществом.[24] Прекрасно известно, что стандартные алгоритмы NIST[25] не лишены недостатков. Например, недавно стало очевидно, что алгоритм генерации случайных чисел PNRG (его версия, основанная на использовании эллиптических кривых) имеет определённый изъян и может содержать потенциальную лазейку [68]. Если смотреть с более широкой перспективы, организации, занимающиеся стандартизацией, такие как NIST, способствуют развитию криптографической монокультуры, создают условия для централизации. Великолепным примером может служить то, как Агентство национальной безопасности США (NSA) использовало своё влияние на NIST, чтобы ослабить международный криптографический стандарт [18].

Кривая Ed25519 не запатентована (в работе [79] обсуждается эта тема), и команда, стоящая за её созданием, занималась разработкой и адаптацией базовых криптографических алгоритмов, ориентируясь на их эффективность [40].

src/crypto/crypto_ops_builder/

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 будет десятичным числом 76 ($l$ занимает 253 бита):[26]

$$2^3 \cdot 7237005577332262213973186563042994240857116359379907606001950938285454250989$$

src/ringct/rctOps.h `curveOrder()`

### 2.4.1 Двоичное представление

Элементы $\mathbb{F}_{2^{255}-19}$ являются 256-битными целыми числами, поэтому 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.

[24] Даже если с кривой не связано никаких вопросов относительно её криптографической безопасности, существует вероятность того, что человеку/организации, создавшей её, будет известен некий секретный недостаток, который будет всплывать только в очень редких случаях. Такой человек может случайным образом генерировать множество кривых, чтобы найти ту, у которой будет скрытая уязвимость, а известные слабые места будут отсутствовать. Если параметры кривой потребуют разумных объяснений, будет ещё сложнее найти уязвимые кривые, которые будут приняты криптографическим сообществом. Кривая Ed25519 известна как «крайне стабильная» кривая, а это означает, что процесс её генерирования имеет полное объяснение. [120]

[25] Национальный институт стандартов и технологий США, https://www.nist.gov/

[26] Это означает, что приватные ключи EC, в случае с Ed25519, занимают 253 бита.

### 2.4.2 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. Мы не станем углубляться в математические подробности, необходимые для того, чтобы объяснить это, но мы можем кратко показать, как это работает [74]. . Процесс сжатия точек кривой Ed25519 впервые был описан в работе [39], в то время как сама концепция впервые была изложена в работе [94].

Данная схема сжатия точек основана на преобразовании уравнения скрученной кривой Эдвардса (с допуском $a = -1$, что действительно для Monero): $x^2 = (y^2 - 1)/(dy^2 + 1)$,[27] которое демонстрирует, что у $x$ для каждого $y$ может быть только два возможных значения (+ или −). Элементы поля $x$ и $y$ вычисляются $(\mathrm{mod}\ q)$, поэтому фактические отрицательные значения отсутствуют. Тем не менее $(\mathrm{mod}\ q)$ от $\breve{}x$ изменит значение между нечётными и чётными, так как $q$ является нечётным. Например, $3\ (\mathrm{mod}\ 5) = 3$, $-3\ (\mathrm{mod}\ 5) = 2$. Другими словами, элементам поля $x$ и $\breve{}x$ присваиваются различные чётные/нечётные значения.

Если нам известно, что $x$ имеет чётное значение, но при заданном значении $y$ преобразованное уравнение кривой даёт нечётное число, 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 $(x, y)$.

**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.

src/crypto/crypto_ops_builder/ref10CommentedCombined/ge_tobytes.c

**Дешифровка**

1. Берём сжатую точку $y'$, а затем копируем её наиболее значимый бит в контрольный бит чётности $b$, перед тем как присвоить ему нулевое значение. Теперь это снова будет оригинальный $y$.
2. Допустим, $u = y^2 - 1\ (\mathrm{mod}\ q)$, а $v = dy^2 + 1\ (\mathrm{mod}\ q)$. Это означает $x^2 = u/v\ (\mathrm{mod}\ q)$.
3. Вычисляем[28] $z = uv^3(uv^7)^{(q-5)/8}\ (\mathrm{mod}\ q)$.
   (a) Если $vz^2 = u\ (\mathrm{mod}\ q)$, значит $x' = z$.
   (b) Если $vz^2 = -u\ (\mathrm{mod}\ q)$, вычисляем $x' = z * 2^{(q-1)/4}\ (\mathrm{mod}\ q)$.
4. Используя контрольный бит чётности $b$ полученный при выполнении первого этапа, если $b \neq$ наименее значимому биту $x'$, получаем $x = -x'\ (\mathrm{mod}\ q)$, в противном случае получаем $x = x'$.

ge_frombytes.c

[27] В данном случае $d = -\frac{121665}{121666}$.
[28] Так как $q = 2^{255} - 19 \equiv 5\ (\mathrm{mod}\ 8)$, $(q-5)/8$ и $(q-1)/4$ являются целыми числами.

5. Получаем несжатую точку $(x, y)$.

Варианты реализации Ed25519 (как в случае с Monero) обычно используют генератор $G = (x, 4/5)$ [39], где $x$ является «чётным», или $b = 0$, вариант, основанный на развёртывании точки $y = 4/5 \pmod q$.

### 2.4.3 Алгоритм подписи EdDSA

Бернштейном и его командой был разработан ряд базовых алгоритмов, основанных на кривой Ed25519.[29] Для примера нами описана предельно оптимизированная и безопасная альтернатива схеме подписи ECDSA, которая, со слов её авторов, позволяет создавать более 100 000 подписей в секунду, используя обычный потребительский процессор Intel Xeon [39]. Алгоритм также описан в Internet RFC8032 [75]. . Следует отметить, что эта схема имеет много общего со схемой подписи Шнорра.

Помимо прочего, вместо постоянной генерации случайных целых чисел в данном случае используется значение хеша, выведенное на основе приватного ключа подписанта 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 секретным или непрогнозируемым областям памяти, что позволяет не избежать так называемых *атак кэша по времени* [39].

В этой работе исключительно для наглядности нами кратко описаны этапы алгоритма. Полное описание и пример реализации на языке Python можно найти в работе [75].

#### Подпись

1. Допустим, $h_k$ является хешем $\mathcal{H}(k)$ приватного ключа $k$ подписанта. Вычисляем $\alpha$ как $\alpha = \mathcal{H}(h_k, \mathfrak{m})$ хешированного приватного ключа и сообщения. В зависимости от варианта реализации, $\mathfrak{m}$ может являться фактическим сообщением или его хешем [75].
2. Вычисляем $\alpha G$ и запрос $ch = \mathcal{H}([\alpha G], K, \mathfrak{m})$.
3. Вычисляем ответ $r = \alpha + ch \cdot k$.
4. Подпись будет представлена парой $(\alpha G, r)$.

[29] В работе [40] описана группа эффективных операций со скрученной эллиптической кривой Эдвардса (то есть добавление точки, удвоение, смешанное добавление и т.д.). В работе [36] приводятся эффективные модульные арифметические операции.

src/crypto/crypto_ops_builder/ref10CommentedCombined/

#### Верификация

Верификация производится следующим образом

1. Вычисляем $ch' = \mathcal{H}([\alpha G], K, \mathfrak{m})$.
2. Если равенство $2^c rG \stackrel{?}{=} 2^c \alpha G + 2^c ch' * K$ соблюдается, значит, подпись является действительной.

Член уравнения $2^c$ взят из общей формы Бернштейна алгоритма EdSDA [39]. . Согласно этой работе, хотя этого и не требуется для адекватной верификации, удаление $2^c$ усиливает уравнения.

Публичный ключ $K$ может быть любой точкой на кривой EC, но мы хотим использовать только те точки, которые входят в подгруппу генератора $G$. Умножение на кофактор $2^c$ гарантирует, что все точки будут находиться в подгруппе. Как вариант, верификаторы могут проверять соответствие $lK \stackrel{?}{=} 0$, которое будет таковым только в том случае, если $K$ будет принадлежать подгруппе. Нам неизвестно, сколько уязвимостей позволяют нейтрализовать эти меры предосторожности, однако мы ещё увидим, насколько последний метод важен для Monero (раздел 3.4).

В случае с этой схемой подписи мы сохраняем одну точку на кривой EC и одну скалярную величину, и у нас получается один публичный ключ EC.

#### Почему это работает

$$\begin{aligned} 2^c rG &= 2^c(\alpha + \mathcal{H}([\alpha G], K, \mathfrak{m}) \cdot k) \cdot G \\ &= 2^c \alpha G + 2^c \mathcal{H}([\alpha G], K, \mathfrak{m}) \cdot K \end{aligned}$$

#### Двоичное представление

По умолчанию подписи EdDSA для представления требуется $64+32$ байта для точки EC $\alpha G$ и скалярной величины $r$. Тем не менее RFC8032 предполагает, что точка $\alpha G$ является сжатой, что снижает требования к занимаемому пространству до $32 + 32$ байт. Мы также включаем сюда публичный ключ $K$, в результате чего получаем всего $32 + 32 + 32$ байт.

## 2.5 Двоичный оператор XOR

Двоичный оператор XOR является полезным инструментом, о котором ещё зайдёт речь в разделах 4.4 и 5.3. Он берёт два аргумента и возвращает истинный, если один из них, но не оба, является таковым [22]. Таблица истинности выглядит следующим образом:

| A | B | A XOR B |
|---|---|---|
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | F |

В контексте компьютерной науки операция XOR эквивалентна битовому сложению по модулю 2. Например, XOR для двух пар битов:

$$\begin{aligned}\mathrm{XOR}(\{1,1\},\{1,0\}) &= \{1+1, 1+0\} \pmod 2 \\ &= \{0,1\}\end{aligned}$$

Каждая из них также производит $\{0,1\}$: $\mathrm{XOR}(\{1,0\},\{1,1\})$, $\mathrm{XOR}(\{0,0\},\{0,1\})$ и $\mathrm{XOR}(\{0,1\},\{0,0\})$. Для входов XOR с $b$ бит существует $2^{\mathrm{b}}-1$ других комбинаций входов, который дадут тот же выход. Это означает, что если $C = \mathrm{XOR}(A,B)$ и вход $A \in_R \{0,...,2^{\mathrm{b}-1}\}$, наблюдатель, которому стало известно значение $C$, не получит никакой информации о $B$.

В то же самое время любой, корму известны два элемента из трёх $\{A,B,C\}$, где $C = \mathrm{XOR}(A,B)$, сможет вычислить третий элемент, например, $A = \mathrm{XOR}(B,C)$. XOR показывает, являются ли два элемента одинаковыми или разными, поэтому знания $C$ и $B$ достаточно для выделения $A$. Подробное изучение таблицы истинности вскрывает эту крайне важную особенность.[30]

[30] Одним из интересных вариантов применения XOR (не связанным с Monero) является замена двух разрядных регистров без использования третьего регистра. Для обозначения операции XOR мы используем символ $\oplus$ to indicate an XOR operation. $A \oplus A = 0$, и после трёх операций XOR между двумя регистрами: $\{A,B\} \to \{[A \oplus B], B\} \to \{[A \oplus B], B \oplus [A \oplus B]\} = \{[A \oplus B], A \oplus 0\} = \{[A \oplus B], A\} \to \{[A \oplus B] \oplus A, A\} = \{B, A\}$.

# ГЛАВА 3

# Продвинутые подписи Шнорра

В своём базовом варианте подпись Шнорра имеет один подписывающий ключ. Как оказалось, эту ключевую концепцию можно применить для создания целого множества всё более сложных схем подписи. Одна из этих схем, MLSAG, занимает центральное по важности место в протоколе транзакций Monero.

## 3.1 Доказательство знания дискретного логарифма на основе множества «базовых» ключей

Зачастую бывает полезно доказать, что один и тот же приватный ключ использовался для создания различных публичных ключей на основе различных «базовых» ключей. Например, у нас может иметься обычный публичный ключ $kG$ и общий секрет Диффи-Хеллмана $kR$ с публичным ключом какого-либо другого человека (см. подпункт 2.3.3), где базовыми ключами являются $G$ и $R$. Как можно будет скоро увидеть, мы можем доказать знание дискретного логарифма $k$ в $kG$, доказать знание $k$ в $kR$, $u$ доказать, что $k$ является одним и тем же в обоих случаях (при этом $k$ никак не раскрывается).

### Неинтерактивное доказательство

Допустим, у нас есть приватный ключ $k$ и $d$ базовых ключей $\mathcal{J} = \{J_1, ..., J_d\}$. Соответствующими публичными ключами будут $\mathcal{K} = \{K_1, ..., K_d\}$. Создаём доказательство в стиле Шнорра

(см. подпункт 2.3.4) по всем базовым ключам.[1] Допустим, существует хеш-функция $\mathcal{H}_n$, которая распределяется по целым числам в диапазоне от 0 до $l-1$.[2]

1. Генерируем случайное число $\alpha \in_R \mathbb{Z}_l$ и вычисляем для всех $i \in (1, ..., d)$, $\alpha J_i$.
2. Вычисляем запрос:
$$c = \mathcal{H}_n(\mathcal{J}, \mathcal{K}, [\alpha J_1], [\alpha J_2], ..., [\alpha J_d])$$
3. Определяем ответ: $r = \alpha - c * k$.
4. Публикуем подпись $(c, r)$.

### Верификация

Допустим, верификатору известны $\mathcal{J}$ и $\mathcal{K}$, и он делает следующее.

1. Вычисляет запрос:
$$c' = \mathcal{H}(\mathcal{J}, \mathcal{K}, [rJ_1 + c * K_1], [rJ_2 + c * K_2], ..., [rJ_d + c * K_d])$$
2. Если $c = c'$, значит, подписанту должен быть известен дискретный логарифм по всем базовым ключам, и в каждом случае это будет один и тот же дискретный логарифм (как и всегда, за исключением ничтожной вероятности).

### Почему это работает

Если бы вместо $d$ базовых ключей был всего один, очевидно, что доказательство было бы тем же, что и наше оригинальное доказательство Шнорра (см. подпункт 2.3.4). Мы можем представить каждый базовый ключ отдельно, и это позволит нам увидеть, что доказательство по множеству базовых ключей является просто совокупностью доказательств Шнорра, объединённых в одну группу. Более того, при использовании только одного запроса и ответа для всех этих доказательств, у них будет один и тот же дискретный логарифм $k$. Чтобы получить отдельный ответ, который будет работать в случае с множеством ключей, запрос должен быть известен до того, как $\alpha$ будет определена для каждого ключа, но $c$ при этом должна являться функцией $\alpha$!

[1] Несмотря на то, что мы говорим «доказательство», можно просто создать подпись, включив сообщение $\mathfrak{m}$ в хеш запроса. В этом контексте термины являются взаимозаменимыми.

[2] В случае с Monero хеш-функция $\mathcal{H}_n(x) = \text{sc\_reduce32}(\mathit{Keccak}(x))$, где *Keccak* будет основой SHA3, а sc_reduce32() определит 256-битный результат в диапазоне от 0 до $l-1$ (хотя, по сути, результат должен находиться в диапазоне от 1 до $l-1$).

## 3.2 Множество приватных ключей в одном доказательстве

Во многом, как и в случае с доказательствами по множеству базовых ключей, мы можем объединить доказательства Шнорра, использующие различные приватные ключи. Это позволяет доказать, что нам известны все приватные ключи для набора публичных ключей, и сокращает место, необходимое для хранения, так как для всех доказательств требуется всего один запрос.

### Неинтерактивное доказательство

Допустим, у нас есть $d$ базовых ключей $k_1, ..., k_d$ и базовые ключи $\mathcal{J} = \{J_1, ..., J_d\}$.[3] Соответствующими публичными ключами будут $\mathcal{K} = \{K_1, ..., K_d\}$. Создаём доказательство в стиле Шнорра одновременно для всех ключей.

1. Генерируем случайные числа $\alpha_i \in_R \mathbb{Z}_l$ для всех $i \in (1, ..., d)$ и вычисляем все $\alpha_i J_i$.
2. Вычисляем запрос:
$$c = \mathcal{H}_n(\mathcal{J}, \mathcal{K}, [\alpha_1 J_1], [\alpha_2 J_2], ..., [\alpha_d J_d])$$
3. Определяем каждый ответ: $r_i = \alpha_i - c * k_i$.
4. Публикуем подпись $(c, r_1, ..., r_d)$.

### Верификация

Допустим, верификатору известны $\mathcal{J}$ и $\mathcal{K}$, и он делает следующее.

1. Вычисляет запрос:
$$c' = \mathcal{H}(\mathcal{J}, \mathcal{K}, [r_1 J_1 + c * K_1], [r_2 J_2 + c * K_2], ..., [r_d J_d + c * K_d])$$
2. Если $c = c'$, значит, подписанту должны быть известны приватные ключи для всех публичных ключей в $\mathcal{K}$ (за исключением ничтожной вероятности).

[3] В данном случае нет никакой причины, по которой $\mathcal{J}$ не мог бы содержать дублирующие друг друга базовые ключи или же был одинаковым для всех базовых ключей (например, $G$). В случае с доказательствами по множеству базовых ключей «дубликаты» будут избыточными, но здесь мы имеем дело с другими приватными ключами.

## 3.3 Подписи спонтанной анонимной группы (SAG)

Групповые подписи являются способом доказательства того, что подписант принадлежит к группе, без его идентификации. Изначально (см. Чаум (Chaum) [49]) 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 внутри компаний.

Лю *и др.* (Liu et al.) в работе [85] была представлена более интересная схема, основанная на исследованиях Ривеста *и др.* (Rivest et al.) [119]. Авторами был подробно изложен алгоритм формирования групповых подписей под названием LSAG, характеризуемый тремя свойствами: *анонимность, связываемость* и *спонтанность*. Для простоты понимания концепции в данном случае нами рассматривается SAG, несвязываемая версия LSAG. Рассмотрение идеи связываемости мы оставляем для последующих разделов.

Схемы, обладающие свойствами анонимности и спонтанности, как правило, называются «кольцевыми подписями». В контексте Monero в конечном счёте они обеспечивают возможность проведения транзакций с сокрытием подписанта и невозможностью подделки, что делает потоки валюты по большей части неотслеживаемыми.

### Подпись

Кольцевые подписи состоят из кольца и подписи. Каждое *кольцо* является набором публичных ключей, один из которых принадлежит подписанту, а остальные не связаны между собой. *Подпись* генерируется при помощи этого кольца, состоящего из ключей, и верификатор никогда не сможет точно сказать, кто из участников кольца является действительным подписантом.

Нашу схему в стиле Шнорра, описанную в подпункте 2.3.5, можно рассматривать в качестве кольцевой подписи с одним ключом. Мы приходим к использованию двух ключей путём создания ложного $r$ (вместо определения $r'$) и создания нового запроса для определения $r$.

Допустим, $\mathfrak{m}$ является подписываемым сообщением, $\mathcal{R} = \{K_1, K_2, ..., K_n\}$ является набором определённых публичных ключей (группой/кольцом), а $k_\pi$ является приватным ключом подписанта, соответствующим его публичному ключу $K_\pi \in \mathcal{R}$, где $\pi$ является секретным индексом.

1. Генерируем случайное число $\alpha \in_R \mathbb{Z}_l$ и ложные ответы $r_i \in_R \mathbb{Z}_l$ для $i \in \{1, 2, ..., n\}$, но за исключением $i = \pi$.

2. Вычисляем
$$c_{\pi+1} = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [\alpha G])$$
3. Для $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$, заменив $n + 1 \to 1$, вычисляем
$$c_{i+1} = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_i G + c_i K_i])$$
4. Определяем такой реальный ответ $r_\pi$, чтобы $\alpha = r_\pi + c_\pi k_\pi \pmod{l}$.

Кольцевая подпись содержит подпись $\sigma(\mathfrak{m}) = (c_1, r_1, ..., r_n)$ и кольцо $\mathcal{R}$.

### Верификация

Процесс верификации означает доказательство того, что $\sigma(\mathfrak{m})$ является действительной подписью, созданной с использованием приватного ключа, соответствующего публичному ключу в кольце $\mathcal{R}$ (при этом не обязательно известно, какому именно), и производится следующим образом:

1. Для $i = 1, 2, ..., n$, заменив $n + 1 \to 1$, итеративно вычисляем
$$c'_{i+1} = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_i G + c_i K_i])$$
2. Если $c_1 = c'_1$, то подпись является действительной. Следует отметить, что $c'_1$ вычисляется в последнюю очередь.

В рамках этой схемы мы сохраняем $(1+n)$ целых чисел и используем $n$ публичных ключей.

### 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:

1. Генерируем случайные числа: $\alpha$, $r_1$, $r_3$
2. Вводим цикл подписи:
$$c_3 = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [\alpha G])$$
3. Производим итерацию:
$$c_1 = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_3 G + c_3 K_3])$$
$$c_2 = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_1 G + c_1 K_1])$$
4. Замыкаем цикл ответом: $r_2 = \alpha - c_2 k_2 \pmod{l}$

Мы можем заменить $\alpha$ на $c_3$, чтобы понять, откуда взялось слово «кольцо»:

$$c_3 = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [(r_2 + c_2 k_2)G])$$
$$c_3 = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_2 G + c_2 K_2\ ])$$

Затем производится верификация с использованием $\mathcal{R}$ и $\sigma(\mathfrak{m}) = (c_1, r_1, r_2, r_3)$:

1. Используем $r_1$ и $c_1$ для вычисления
$$c_2' = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_1 G + c_1 K_1])$$
2. С момента создания подписи мы видим, что $c_2' = c_2$. Используя $r_2$ и $c_2'$ , вычисляем
$$c_3' = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_2 G + c_2' K_2])$$
3. Заменив $c_2$ на $c_2'$, мы увидим, что $c_3' = c_3$. Используя $r_3$ и $c_3'$ , получаем
$$c_1' = \mathcal{H}_n(\mathcal{R}, \mathfrak{m}, [r_3 G + c_3' K_3])$$

В данном случае ничего удивительного: если заменить $c_3$ на $c_3'$, то $c_1' = c_1$.

## 3.4 Связываемые подписи спонтанной анонимной группы Бэка (bLSAG)

Кольцевые подписи, рассматриваемые здесь, демонстрируют несколько свойств, которые будут полезны при создании конфиденциальных транзакций.[4] Необходимо отметить, что как понятие «сокрытия подписанта», так и «невозможности подделки» также относятся к подписям SAG.

**Сокрытие подписанта** Наблюдатель должен быть в состоянии определить, что подписант является участником кольца, но не должен знать, каким именно[5] Monero использует это для маскировки источника средств при проведении каждой транзакции.

[4] Следует помнить о том, что все устойчивые схемы подписей имеют модели безопасности, обладающие различными свойствами. Свойства, упомянутые здесь, вероятно, будут наиболее важными с точки зрения понимания предназначения кольцевых подписей Monero, но они не могут служить для полного описания свойств связываемой кольцевой подписи.

[5] Анонимность в случае совершения какого-либо действия с точки зрения «анонимной группы», которая включает в себя всех людей, которые могли совершить такое действие. Самой большой анонимной группой является «человечество», а в случае с Monero это размер кольца или, например, так называемый «уровень смешивания» $v$ плюс реальный подписант. Термин «смешивание» подразумевает количество ложных участников в каждой кольцевой подписи. Если значение смешивания составляет $v = 4$, значит, присутствует 5 возможных подписантов. Расширение анонимной группы усложняет процесс определения реальных участников транзакции.

**Связываемость** Если приватный ключ используется для того, чтобы подписать два различных сообщения, то эти сообщения становятся связанными.[6] Как будет показано далее, это свойство помогает предотвратить атаки, связанные с двойной тратой Monero (за исключением ничтожной вероятности).

**Невозможность подделки** Никакой злоумышленник не сможет сгенерировать фальшивую подпись, кроме как с ничтожной вероятностью.[7] Это не позволяет похитить Monero тем, кто не владеет соответствующими приватными ключами.

В соответствии со схемой подписи LSAG [85] владелец приватного ключа может создавать одну анонимную несвязанную подпись на кольцо.[8] В этом разделе нами предлагается улучшенная версия алгоритма LSAG, где свойство связываемости не зависит от ложных участников кольца.[9]

Модификация была предложена в работе [108], основанной на публикации Адама Бэка (Adam Back) [35], касающейся алгоритма кольцевой подписи CryptoNote [136] (ранее использовался Monero, но теперь от него отказались; см. подпункт 8.1.2), в основу которого, в свою очередь, легла работа Фуджисаки (Fujisaki) и Сузуки (Suzuki) [65].

## Подпись

Как и в случае с SAG, допустим, что $\mathfrak{m}$ является подписываемым сообщением, $\mathcal{R} = \{K_1, K_2, ..., K_n\}$ является набором определённых публичных ключей, а $k_\pi$ является приватным ключом подписанта, соответствующим его публичному ключу $K_\pi \in \mathcal{R}$, где $\pi$ является секретным индексом. Также допустим наличие хеш-функции $\mathcal{H}_p$, отвечающей за распределение точек на кривой EC.[10,11]

1. Вычисляем образ ключа $\tilde{K} = k_\pi \mathcal{H}_p(K_\pi)$.[12]

[6] 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.

[7] Определенные схемы кольцевых подписей, включая ту, что 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. [108] и [85].

[8] В рамках схемы LSAG связываемость реализуется в случае только с теми подписями, которые используют кольца с теми же участниками и в том же самом порядке, то есть это должно быть «абсолютно то же самое кольцо». Фактически это «одна анонимная подпись на кольцо». Подписи могут быть связаны, даже если были созданы для разных сообщений.

[9] Схема LSAG рассматривалась в первой редакции данного отчёта. [33]

[10] Совершенно неважно, буду точки $\mathcal{H}_p$ сжатыми или нет. Они всегда могут быть развёрнуты.

[11] Monero использует хеш-функцию, которая выдаёт точки кривой напрямую, а не путём вычисления какого-либо целого числа, которое затем умножается на $G$. $\mathcal{H}_p$ была бы разбита, если бы кто-то нашёл способ найти такое $n_x$, чтобы $n_x G = \mathcal{H}_p(x)$. Описание алгоритма приводится в работе [107]. В соответствии с белой книгой CryptoNote [136] изначально он был представлен в работе [133]. src/ringct/rctOps.cpp `hash_to_p3()`

[12] В случае с Monero для образов ключей важно использовать функцию преобразования хеша в точку вместо другой базовой точки. Таким образом, линейность не приведёт к связыванию подписей, созданных по одному и тому же адресу (даже в том случае, если они были созданы для разных одноразовых адресов). См. [136], стр. 18.

2. Генерируем случайное число $\alpha \in_R \mathbb{Z}_l$ и случайные числа $r_i \in_R \mathbb{Z}_l$ для $i \in \{1, 2, ..., n\}$, но за исключением $i = \pi$.

3. Вычисляем
$$c_{\pi+1} = \mathcal{H}_n(\mathfrak{m}, [\alpha G], [\alpha \mathcal{H}_p(K_\pi)])$$

4. Для $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$ заменив $n + 1 \to 1$, вычисляем
$$c_{i+1} = \mathcal{H}_n(\mathfrak{m}, [r_i G + c_i K_i], [r_i \mathcal{H}_p(K_i) + c_i \tilde{K}])$$

5. Определяем $r_\pi = \alpha - c_\pi k_\pi \pmod{l}$.

Кольцевой подписью будет $\sigma(\mathfrak{m}) = (c_1, r_1, ..., r_n)$ с образом ключа $\tilde{K}$ и кольцом $\mathcal{R}$.

### Верификация

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

1. Проверяем $l\tilde{K} \stackrel{?}{=} 0$.

2. Для $i = 1, 2, ..., n$, заменив $n + 1 \to 1$, итеративно вычисляем
$$c'_{i+1} = \mathcal{H}_n(\mathfrak{m}, [r_i G + c_i K_i], [r_i \mathcal{H}_p(K_i) + c_i \tilde{K}])$$

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

В рамках этой схемы мы сохраняем $(1+n)$ целых чисел, имеем один образ ключа EC и используем $n$ публичных ключей.

Нам необходимо проверить $l\tilde{K} \stackrel{?}{=} 0$, поскольку есть возможность добавить точку EC из подгруппы с размером $h$ (кофактор) в $\tilde{K}$ и, если все $c_i$ будут кратными $h$ (чего можно достичь при помощи автоматизированного последовательного приближения с использованием различных значений $\alpha$ и $r_i$), создать $h$ несвязанных действительных подписей при помощи того же кольца и подписывающего ключа.[13] Причина заключается в том, что точка EC, помноженная на порядок своей подгруппы, будет равна нулю.[14]

src/cryptonote_core/cryptonote_core.cpp `check_tx_inputs_key-images_domain()`

[13] Нас не интересуют точки из других подгрупп, поскольку выход $\mathcal{H}_n$ ограничивается до $\mathbb{Z}_l$. Для EC порядка $N = hl$ все делители $N$ (а следовательно, и возможные подгруппы) будут либо кратными $l$ (простому числу), либо будут делителями $h$.

[14] На ранних этапах истории Monero это не проверялось. К счастью, этим никто не воспользовался до того момента, когда в апреле 2017 были внесены соответствующие исправления (была реализована версия v5 протокола) [63].

Для ясности: при наличии точки $K$ в подгруппе порядка $l$, некоторой точки $K^h$ порядка $h$ и целого числа $c$, делимого на $h$:

$$\begin{aligned} c * (K + K^h) &= cK + cK^h \\ &= cK + 0 \end{aligned}$$

Мы демонстрируем правильность (то есть «как это работает») подобно тому, как это делали в случае с более простой схемой подписи SAG.

В нашем описании мы пытаемся придерживаться оригинального объяснения bLSAG, которое не включает $\mathcal{R}$ в хеш, используемый для вычисления $c_i$. Включение ключей в хеш известно как «добавление префикса ключа». Недавние исследования [77] позволяют предположить, что в этом нет необходимости, хотя добавление префикса является стандартной практикой при использовании подобных схем подписи (LSAG использует добавление префикса ключа).

### Связываемость

При наличии двух действительных подписей, отличающихся некоторым образом (например, имеющих разные ложные ответы, различные сообщения, различное общее количество участников кольца)

$$\begin{aligned} \sigma(\mathfrak{m}) &= (c_1, r_1, ..., r_n) \text{ with } \tilde{K}, \text{ and} \\ \sigma'(\mathfrak{m}') &= (c'_1, r'_1, ..., r'_{n'}) \text{ with } \tilde{K}', \end{aligned}$$

и если $\tilde{K} = \tilde{K}'$, значит, в основе обеих подписей лежит один и тот же приватный ключ.

Несмотря на то, что внешний наблюдатель сможет связать $\sigma$ и $\sigma'$, совсем не обязательно ему будет известно, какое $K_i$ в $\mathcal{R}$ или $\mathcal{R}'$ было виновным, если между ними был только один общий ключ. При наличии более одного общего участника кольца наблюдателю останется только решить DLP или каким-либо образом провести аудит колец (например, узнать все $k_i$, где $i \neq \pi$, или узнать $k_\pi$).[15]

## 3.5 Многоуровневые связываемые подписи спонтанной анонимной группы (MLSAG)

Для того чтобы подписать транзакцию, понадобится множество приватных ключей. В работе [108], Шена Нётера *и др.* описывают многоуровневое обобщение схемы подписей bLSAG, которое может применяться в случае наличия ряда $n \cdot m$ ключей, то есть набора

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

[15] LSAG, как и bLSAG, не поддаётся поделке, поэтому никакой злоумышленник не сможет сгенерировать действительную кольцевую подпись, не зная приватного ключа. Если он придумает фальшивый образ $\tilde{K}$ и запустит вычисление подписи, используя $c_{\pi+1}$, то, не зная $k_\pi$, он не сможет вычислить число $r_\pi = \alpha - c_\pi k_\pi$, которое позволит получить $[r_\pi G + c_\pi K_\pi] = \alpha G$. Верификатор отклонит его подпись. В работе Лю *и др.* доказано, что создание подделок, которые могли бы пройти верификацию, крайне маловероятно [85].

для которого нам известно $m$ приватных ключей $\{k_{\pi,j}\}$, соответствующих поднабору $\{K_{\pi,j}\}$, для некоторого индекса $i = \pi$. Такой алгоритм удовлетворил бы наши потребности, если бы мы обобщили 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. Вычисляем образы ключей $\tilde{K}_j = k_{\pi,j}\mathcal{H}_p(K_{\pi,j})$ для всех $j \in \{1, 2, ..., m\}$.
2. Генерируем случайные числа $\alpha_j \in_R \mathbb{Z}_l$ и $r_{i,j} \in_R \mathbb{Z}_l$ для $i \in \{1, 2, ..., n\}$ (за исключением $i = \pi$) и $j \in \{1, 2, ..., m\}$.
3. Вычисляем[16]
$$c_{\pi+1} = \mathcal{H}_n(\mathfrak{m}, [\alpha_1 G], [\alpha_1 \mathcal{H}_p(K_{\pi,1})], ..., [\alpha_m G], [\alpha_m \mathcal{H}_p(K_{\pi,m})])$$
4. Для $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$, заменив $n + 1 \to 1$, вычисляем
$c_{i+1} = \mathcal{H}_n(\mathfrak{m}, [r_{i,1}G + c_i K_{i,1}], [r_{i,1}\mathcal{H}_p(K_{i,1}) + c_i \tilde{K}_1], ..., [r_{i,m}G + c_i K_{i,m}], [r_{i,m}\mathcal{H}_p(K_{i,m}) + c_i \tilde{K}_m])$
5. Определяем все $r_{\pi,j} = \alpha_j - c_\pi k_{\pi,j} \pmod{l}$.

Кольцевой подписью будет $\sigma(\mathfrak{m}) = (c_1, r_{1,1}, ..., r_{1,m}, ..., r_{n,1}, ..., r_{n,m})$ с образами ключей $(\tilde{K}_1, ..., \tilde{K}_m)$.

### Верификация

Верификация подписи производится следующим образом:

1. Для всех $j \in \{1, ..., m\}$ проверяем $l\tilde{K}_j \stackrel{?}{=} 0$.
2. Для $i = 1, ..., n$, заменив $n + 1 \to 1$, вычисляем
$c'_{i+1} = \mathcal{H}_n(\mathfrak{m}, [r_{i,1}G + c_i K_{i,1}], [r_{i,1}\mathcal{H}_p(K_{i,1}) + c_i \tilde{K}_1], ..., [r_{i,m}G + c_i K_{i,m}], [r_{i,m}\mathcal{H}_p(K_{i,m}) + c_i \tilde{K}_m])$
3. Если $c_1 = c'_1$, то подпись является действительной.

[16] MLSAG-подпись Monero использует добавление префикса ключа. Каждый запрос содержит чётко заданные публичные ключи, подобные этому (добавление элементов $K$ отсутствует в bLSAG; в подписанные сообщения включаются образы ключей): src/ringct/rctSigs.cpp `MLSAG_Gen()`
$$c_{\pi+1} = \mathcal{H}_n(\mathfrak{m}, K_{\pi,1}, [\alpha_1 G], [\alpha_1 \mathcal{H}_p(K_{\pi,1})], ..., K_{\pi,m}, [\alpha_m G], [\alpha_m \mathcal{H}_p(K_{\pi,m})])$$

### Почему это работает

Как и в случае с оригинальным алгоритмом SAG, мы видим, что:

- если $i \neq \pi$, то очевидно значения $c'_{i+1}$ вычисляются, как описано в алгоритме подписи;
- если $i = \pi$, то, $r_{\pi,j} = \alpha_j - c_\pi k_{\pi,j}$ закрывает цикл,

$$r_{\pi,j}G + c_\pi K_{\pi,j} = (\alpha_j - c_\pi k_{\pi,j})G + c_\pi K_{\pi,j} = \alpha_j G$$

и

$$r_{\pi,j}\mathcal{H}_p(K_{\pi,j}) + c_\pi \tilde{K}_j = (\alpha_j - c_\pi k_{\pi,j})\mathcal{H}_p(K_{\pi,j}) + c_\pi \tilde{K}_j = \alpha_j \mathcal{H}_p(K_{\pi,j})$$

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

### Связываемость

Если для создания какой-либо подписи повторно используется приватный ключ $k_{\pi,j}$ , соответствующий образ ключа $\tilde{K}_j$, передаваемый вместе с подписью, выявит его. Это соответствует нашему обобщённому определению связываемости.[17]

### Требования к занимаемому месту

В рамках этой схемы мы сохраняем $(1+m*n)$ целых чисел, имеем $m$ образов ключей EC и используем $m*n$ публичных ключей.

## 3.6 Компактные связываемые подписи спонтанной анонимной группы (CLSAG)

Подписи CLSAG [67][18] находятся где-то между bLSAG и MLSAG. Предположим, у вас имеется «первичный» ключ, а с ним связано несколько «вспомогательных» ключей. Важно доказать знание всех приватных ключей, но свойство связываемости применимо только к первичному. Такое ограничение по связываемости позволяет создавать меньшие по размеру и более быстрые подписи, чем в случае со схемой MLSAG.

Как и в случае с MLSAG у нас имеется набор из $n \cdot m$ ключей (где $n$ является размером кольца, а $m$ — количеством подписывающих ключей), а первичные ключи находятся по индексу 1.

[17] Как и в случае с bLSAG, связанные подписи MLSAG не указывают, какой публичный ключ использовался для подписания. Тем не менее, если подписи в подциклах связующего образа ключа имеют только один общий ключ, виновник становится очевидным. Если виновник выявлен, все остальные участники обеих подписей раскрываются, так как они все используют общие индексы виновника.

[18] В основе данного раздела лежит работа, являющаяся препринтом, который будет вскоре окончательно оформлен для внешней редакции. Схема CLSAG в перспективе может заменить MLSAG с выходом новых версий протокола, но пока она не реализована, и, возможно, этого и не произойдёт в будущем.

Другими словами, существует $n$ первичных ключей, и такой ключ $\pi^{\text{th}}$ и его вспомогательные ключи будут использоваться для подписания.

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

Нам известны приватные ключи $\{k_{\pi,j}\}$, соответствующие поднабору $\{K_{\pi,j}\}$, для некоторого индекса $i = \pi$.

### Подпись

1. Вычисляем образы ключей для $\tilde{K_j} = k_{\pi,j}\mathcal{H}_p(K_{\pi,1})$ всех $j \in \{1, 2, ..., m\}$. Следует отметить, что базовый ключ всегда будет одним и тем же, поэтому образы ключей, где $j > 1$, будут «образами вспомогательных ключей». Для простоты обозначения мы называем их $\tilde{K}_j$.

2. Генерируем случайные числа $\alpha \in_R \mathbb{Z}_l$ и $r_i \in_R \mathbb{Z}_l$ для $i \in \{1, 2, ..., n\}$ (за исключением $i = \pi$).

3. Вычисляем совокупные публичные ключи $W_i$ для $i \in \{1, 2, ..., n\}$ и образ совокупного ключа $\tilde{W}$[19]
$$W_i = \sum_{j=1}^{m} \mathcal{H}_n(T_j, \mathcal{R}, \tilde{K}_1, ..., \tilde{K}_m) * K_{i,j}$$
$$\tilde{W} = \sum_{j=1}^{m} \mathcal{H}_n(T_j, \mathcal{R}, \tilde{K}_1, ..., \tilde{K}_m) * \tilde{K}_j$$
где $w_\pi = \sum_j \mathcal{H}_n(T_j, ...) * k_{\pi,j}$ является совокупным приватным ключом.

4. Вычисляем
$$c_{\pi+1} = \mathcal{H}_n(T_c, \mathcal{R}, \mathfrak{m}, [\alpha G], [\alpha\mathcal{H}_p(K_{\pi,1})])$$

5. Для $i = \pi + 1, \pi + 2, ..., n, 1, 2, ..., \pi - 1$, заменив $n + 1 \to 1$, вычисляем
$$c_{i+1} = \mathcal{H}_n(T_c, \mathcal{R}, \mathfrak{m}, [r_i G + c_i W_i], [r_i\mathcal{H}_p(K_{i,1}) + c_i\tilde{W}])$$

6. Определяем $r_\pi = \alpha - c_\pi w_\pi \pmod{l}$.

Следовательно, $\sigma(\mathfrak{m}) = (c_1, r_1, ..., r_n)$ с образом первичного ключа $\tilde{K}_1$ и вспомогательными образами $(\tilde{K}_2, ..., \tilde{K}_m)$.

[19] В документе CLSAG указано, что для разделения доменов следует использовать разные хеш-функции, что моделируется нами путём добавления тега к каждому хешу в виде префикса [67], например, $T_1$ = “CLSAG_1", $T_c$ = “CLSAG_c" и так далее. Разделённые по доменам хеш-функции дают различные выходы даже при наличии одинаковых входов. Также в этом случае мы используем добавление префиксов ключей (включая $\mathcal{R}$, в который входят все ключи в хеше). Разделение доменов является новой политикой, используемой при разработке Monero, и, вероятнее всего, будет применяться в случае любого использования хеш-функций в будущем (в версиях после v13). Исторические способы использования хеш-функций, вероятно, останутся в прошлом.

### Верификация

Верификация подписи производится следующим образом.

1. Для всех $j \in \{1, ..., m\}$ проверяем $l\tilde{K}_j \stackrel{?}{=} 0$.[20]
2. Вычисляем совокупные публичные ключи $W_i$ для $i \in \{1, 2, ..., n\}$ и совокупный образ ключа $\tilde{W}$
$$W_i = \sum_{j=1}^{m} \mathcal{H}_n(T_j, \mathcal{R}, \tilde{K}_1, ..., \tilde{K}_m) * K_{i,j}$$
$$\tilde{W} = \sum_{j=1}^{m} \mathcal{H}_n(T_j, \mathcal{R}, \tilde{K}_1, ..., \tilde{K}_m) * \tilde{K}_j$$
3. Для $i = 1, ..., n$, заменив $n + 1 \to 1$, вычисляем
$$c_{i+1} = \mathcal{H}_n(T_c, \mathcal{R}, \mathfrak{m}, [r_i G + c_i W_i], [r_i \mathcal{H}_p(K_{i,1}) + c_i \tilde{W}])$$
4. Если $c_1 = c'_1$, то подпись является действительной.

### Почему это работает

Самой большой опасностью в случае с компактными подписями, подобными этой, является аннулирование ключей, если соответствующие образы ключей признаются нелегитимными, но по-прежнему суммируются до легитимного совокупного значения. Вот где в игру вступают агрегирующие коэффициенты $\mathcal{H}_n(T_j, \mathcal{R}, \tilde{K}_1, ..., \tilde{K}_m)$, блокирующие каждый ключ в соответствии с его ожидаемым значением. Мы оставляем отслеживание циклических результатов подделки образов ключей в качестве упражнения для наших читателей (для начала, например, представьте, что эти коэффициенты просто не существуют). Образы вспомогательных ключей являются артефактом доказательства того, что образ первичного ключа является легитимным, так как совокупный приватный ключ $w_\pi$ содержащий все приватные ключи, применяется в отношении базовой точки $\mathcal{H}_p(K_{\pi,1})$.

### Связываемость

Если приватный ключ $k_{\pi,1}$ используется повторно для создания какой-либо подписи, соответствующий образ первичного ключа $\tilde{K}_1$, получаемый с подписью, обнаружит это. Образы вспомогательных ключей игнорируются, поскольку они используются только для облегчения «компактной» части CLSAG.

[20] В случае с Monero мы проверяем только $l * \tilde{K}_1 \stackrel{?}{=} 0$ для образа первичного ключа. Вспомогательные ключи будут сохраняться как $(1/8) * \tilde{K}_j$, а во время верификации умножаться на 8 (см. подпункт 2.3.1), что более эффективно. Противоречивость метода заключается в выборе варианта реализации, поскольку связываемые образы ключей очень важны и нег должны беспорядочно смешиваться, а другой метод использовался с другими версиями протокола.

### Требования к занимаемому месту

Мы сохраняем $(1+n)$ целых чисел, имеем $m$ образов ключей и используем $m * n$ публичных ключей.

# глава 4

# Адреса Monero

Владение цифровой валютой, которая хранится в блокчейне, контролируется посредством так называемых «адресов». На адреса переводятся средства, которые после этого может потратить только владелец такого адреса.[1]

Если точнее, по адресу хранятся «выходы» некоторых транзакций, которые подобны «заметкам», дающим право владельцу адреса потратить некоторую «сумму» денег. В такой заметке будет сказано: «По адресу C теперь имеется 5,3 XMR».

Чтобы потратить имеющийся выход, владелец адреса берёт его в качестве входа в новой транзакции. В эту новую транзакцию также включаются выходы, имеющиеся по другим адресам (или по адресу отправителя, если он того пожелает). Общая сумма по входам транзакции равна общей сумме по выходам, и после того, как они будут потрачены, владелец уже не сможет сделать этого повторно. Кэрол, являющаяся владельцем адреса C, может включить старый выход в новую транзакцию (например, так: «В этой транзакции мне бы хотелось потратить вот этот выход») и добавить заметку: «По адресу B теперь имеется 5,3 XMR».

Балансом для отдельно взятого адреса является сумма средств, имеющихся по нему в соответствии с непотраченными выходами.[2]

[1] За исключением ничтожной вероятности.

[2] Компьютерные приложения, известные как «кошельки», используются для обнаружения и организации выходов, имеющихся по отдельно взятому адресу, а также для хранения приватных ключей, позволяющих проводить новые транзакции, а также передавать эти транзакции в сеть для верификации и последующего включения в блокчейн.

О том, как от внешнего наблюдателя скрывается сумма транзакции, рассказывается в Главе 5. Структура транзакций рассматривается в Главе 6 (а также то, как доказать, что вы тратите выход, которым владеете и который не был потрачен ранее, даже не раскрывая, какой именно выход тратится). А процесс создания денег и роль наблюдателей обсуждаются в Главе 7.

## 4.1 Ключи пользователя

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 2.3.2.

*Адрес* пользователя представлен парой публичных ключей $(K^v, K^s)$. Приватными ключами будет соответствующая пара $(k^v, k^s)$.[3]

src/common/base58.cpp `encode_addr()`

Использование двух наборов ключей делает возможным разделение функций. Обоснованность такого подхода станет очевидной позже в этой главе, а сейчас будем называть приватный ключ $k^v$ *ключом просмотра*, а ключ $k^s$ - *ключом траты*. Пользователь сможет использовать свой ключ просмотра для того, чтобы определить, принадлежит ли выход ему или какому-либо другому адресу, а ключ траты позволит потратить этот выход в транзакции (и ретроактивно вычислить, что выход был потрачен).[4]

## 4.2 Одноразовые адреса

Чтобы получить деньги, пользователь Monero может поделиться своим адресом с другими пользователями, чтобы они использовали его для отправки средств в выходах транзакции.

Адрес никогда не используется напрямую.[5] Вместо этого происходит обмен, как в случае с использованием алгоритма Диффи-Хеллмана. В результате создаётся уникальный *одноразовый адрес* для каждого выхода транзакции, который должен быть выплачен пользователю. Таким образом, даже те внешние наблюдатели, которым известны все публичные адреса

[3] При сообщении адреса другим пользователям он, как правило, шифруется при помощи base-58, схемы кодировки из двоичного в текстовый формат, которая изначально была создана для Bitcoin [28]. Более подробная информация содержится в работе [5].

[4] It В настоящее время, как правило, ключ просмотра $k^v$ равен $\mathcal{H}_n(k^s)$. Это означает, что пользователю просто нужно сохранить свой ключ траты $k^s$, чтобы получить доступ ко всем выходам, которые у него имеются (к их просмотру и трате). Ключ траты обычно представлен последовательностью 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 $k^s = \texttt{sc\_reduce32}(Keccak(a))$ и $k^v = \texttt{sc\_reduce32}(Keccak(Keccak(a)))$. [5]

src/wallet/api/wallet.cpp `create()` wallet2.cpp `generate()` `get_seed()`

[5] Метод, описанный здесь, не является обязательным с точки зрения протокола, а зависит от стандартов, которым следовал разработчик при реализации кошелька. Таким образом, альтернативная версия кошелька может быть реализована в стиле Bitcoin, где адреса получателей включаются в данные транзакций напрямую. Подобный «не соответствующий» кошелёк будет производить выходы транзакций, которые не смогут использовать другие кошельки, а каждый Bitcoin-подобный адрес можно будет использовать только один раз благодаря уникальности образа ключа.

пользователей, не смогут идентифицировать пользователя, получившего какой-либо из выходов определённой транзакции.[6]

Давайте рассмотрим это на примере очень простой транзакции с одним входом и одним выходом — проведении платежа с «нулевой» суммой от Элис к Бобу.

У Боба есть приватные/публичные ключи $(k_B^v, k_B^s)$ и $(K_B^v, K_B^s)$, и Элис известны его публичные ключи (его адрес). Транзакция могла бы происходить следующим образом [136]:

1. Элис генерирует случайное число $r \in_R \mathbb{Z}_l$ и вычисляет одноразовый адрес[7]
$$K^o = \mathcal{H}_n(rK_B^v)G + K_B^s$$
src/crypto/crypto.cpp `derive_public_key()`

2. Элис устанавливает $K^o$ в качестве адресата платежа, добавляет сумму по выходу, равную 0, 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 $k_B^v rG = rK_B^v$. Затем он также может вычислить $K_B'^s = K^o - \mathcal{H}_n(rK_B^v)G$. Как только он увидит, что $K_B'^s = K_B^s$, он будет знать, что выход адресован ему.[8]
src/crypto/crypto.cpp `derive_subaddress_public_key()`

   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 $K_B'^s$ для каждого выхода транзакции в сети и «посмотреть», какие из выходов адресованы Бобу.

4. Одноразовыми ключами для выхода будут:
$$K^o = \mathcal{H}_n(rK_B^v)G + K_B^s = (\mathcal{H}_n(rK_B^v) + k_B^s)G$$
$$k^o = \mathcal{H}_n(rK_B^v) + k_B^s$$

Чтобы потратить выход с нулевой суммой [sic] в новой транзакции, всё, что необходимо сделать Бобу, это доказать, что он является его владельцем, подписав сообщение одноразовым ключом $K^o$. Приватный ключ $k_B^s$ является «ключом траты», так как он необходим для доказательства владения выходом, в то время как $k_B^v$ является «ключом просмотра», так как он может использоваться для поиска выходов, которые Боб может потратить.

Как станет ясно из Главы 6, не зная $k^o$, Элис не сможет вычислить образ ключа, поэтому она никогда не узнает наверняка, потратил ли Боб выход, который она отправила ему.[9]

[6] За исключением ничтожной вероятности.

[7] В случае с Monero каждая копия (даже когда она используется в других частях транзакции) $rk^vG$ умножается на кофактор 8, поэтому в данном случае Элис вычисляет $8 * rK_B^v$, а Боб вычисляет $8 * k_B^v rG$. Мы можем сказать, что это бессмысленно (но это *правило*, которому должны следовать пользователи). Умножение на кофактор гарантирует, что полученная точка будет принадлежать подгруппе $G$, но если $R$ и $K^v$ не будут принадлежать к одной подгруппе, то дискретные логарифмы $r$ и $k^v$ уже нельзя будет использовать для создания общего секрета. src/crypto/crypto.cpp `generate_key_derivation()`

[8] $K_B'^s$ вычисляется при помощи `derive_subaddress_public_key()`, поскольку ключи траты для нормального адреса в таблице поиска ключей траты сохраняются с индексом 0, в то время как подадреса имеют индексы 1, 2... Скоро это будет иметь смысл (см. подпункт 4.3.

[9] Представим, что Элис создаёт две транзакции, каждая из которых будет содержать один и тот же выход

Боб может передать свой ключ просмотра третьей стороне. Такая третья сторона может являться доверенным хранителем, аудитором, налоговым органом и т. д. Кем-то, кто обладает доступом к истории транзакций пользователя, но кто не обладает какими-либо большими правами. Эта третья сторона также сможет расшифровать сумму транзакции, как это описано в подпункте 5.3). В Главе 8 описаны другие способы, при помощи которых Боб мог бы предоставить историю своих транзакций.

### 4.2.1 Транзакции со множеством выходов

Большинство транзакций содержит более одного выхода. При отсутствии иного «сдача» пересылается обратно отправителю.[10,11]

Отправители Monero генерируют только одно случайное значение $r$. Точку кривой $rG$ обычно называют *публичным ключом транзакции*, и она вместе с другими данными транзакции публикуется в блокчейне.

Чтобы гарантировать, что все одноразовые адреса в транзакции с $p$ выходов являются разными, даже в случаях, когда один и тот же адрес применяется дважды, Monero использует индексы выходов. Каждый выход транзакции имеет индекс $t \in \{0, ..., p-1\}$. Приложив это значение к совместно используемым секретным данным перед хешированием, можно гарантировать, что полученные одноразовые адреса будут уникальными:

$$K_t^o = \mathcal{H}_n(rK_t^v, t)G + K_t^s = (\mathcal{H}_n(rK_t^v, t) + k_t^s)G$$
$$k_t^o = \mathcal{H}_n(rK_t^v, t) + k_t^s$$

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_tx_with_tx_key()`

src/crypto/crypto.cpp `derive_public_key()`

## 4.3 Подадреса

Пользователи Monero могут генерировать подадреса на основе каждого адреса [105]. Средства, отправляемые на подадреса, могут быть просмотрены и потрачены при помощи ключей

с одноразового адреса $K^o$, который Боб сможет потратить. Поскольку $K^o$ зависит только от $r$ и $K_B^v$, нет никакой причины, по которой она не смогла бы сделать этого. Боб сможет потратить только один из этих выходов, поскольку у каждого одноразового адреса имеется только один образ ключа. Таким образом, если Бооб не будет осторожным, Элис может обхитрить его. Она может создать транзакцию 1 с большой суммой для Боба, а позже — транзакцию 2 с небольшой суммой для Боба. Если он потратит деньги из транзакции 2, он никогда не сможет потратить деньги из транзакции 1. Фактически никто уже не сможет потратить деньги из транзакции 1. По сути, они «сгорят». Кошельки Monero разработаны так, чтобы игнорировать меньшие суммы в подобной ситуации.

[10] На самом деле после реализации версии v12, от каждой транзакции (не майнинговой) *требуется* наличие двух выходов, даже если это будет означать, что один из выходов буде на нулевую сумму (`HF_VERSION_MIN_2_OUTPUTS`). Это повышает уровень неотличимости транзакций друг от друга за счёт смешивания транзакций с одним выходом с более обычными транзакциями с 2 выходами. До версии v12 программное обеспечение кошелька уже позволяло создавать выходы с нулевой суммой. В соответствии с основным вариантом реализации выходы с нулевой суммой отправляются на случайный адрес.

[11] После реализации Bulletproofs в версии v8 протокола каждая транзакция стала ограничиваться 16 выходами (`BULLETPROOF_MAX_OUTPUTS`). Ранее подобное ограничение отсутствовало, а ограничивался только размер транзакции (в байтах).

src/wallet/wallet2.cpp `transfer_selected_rct()`

просмотра и траты основного адреса. Аналогичным образом: под одной учётной записью пользователя в онлайн банке может иметься множество счетов, 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, 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 [105].[12]

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. Ещё один способ, позволяющий отличить полученные выходы, рассматривается в следующем подпункте.

Боб генерирует свой $i$ подадрес ($i = 1, 2, ...$) на основе своего адреса как пару публичных ключей $(K^{v,i}, K^{s,i})$:[13] src/device/device_default.cpp `get_subaddress_secret_key()`

$$K^{s,i} = K^s + \mathcal{H}_n(k^v, i)G$$
$$K^{v,i} = k^v K^{s,i}$$

Таким образом,

$$K^{v,i} = k^v(k^s + \mathcal{H}_n(k^v, i))G$$
$$K^{s,i} = \quad (k^s + \mathcal{H}_n(k^v, i))G$$

### 4.3.1 Отправка средств на подадрес

Допустим, Элис снова собирается отправить Бобу нулевую сумму, но в этот раз на его подадрес $(K_B^{v,1}, K_B^{s,1})$.

1. Элис генерирует случайное число $r \in_R \mathbb{Z}_l$ и вычисляет одноразовый адрес
$$K^o = \mathcal{H}_n(rK_B^{v,1}, 0)G + K_B^{s,1}$$
2. Элис устанавливает $K^o$ в качестве адресата платежа, добавляет сумму по выходу, равную 0, добавляет значение $rK_B^{s,1}$ к данным транзакции и передаёт их в сеть.

[12] До введения подадресов (были добавлены при обновлении программного обеспечения до версии, соответствующей версии протокола v7 [76]) пользователи могли генерировать только множество обычных адресов. Чтобы просмотреть баланс по каждому из адресов приходилось сканировать отдельную запись в блокчейне. Это было очень неэффективно. После ведения подадресов у пользователей появилась поисковая таблица, содержащая (хешированные) ключи траты. Таким образом, одно сканирование блокчейна занимает столько же времени, что и для 1 подадреса или 10 000 подадресов.

[13] так получается, что хеш подадреса отделён от домена, поэтому на самом деле это $\mathcal{H}_n(T_{sub}, k^v, i)$, где $T_{sub}$ ="SubAddr". В этом документе мы опускаем $T_{sub}$ для краткости.

3. Боб получает данные и видит значения $rK_B^{s,1}$ и $K^o$. Он может вычислить $k_B^v rK_B^{s,1} = rK_B^{v,1}$. Затем он также может вычислить $K_B'^s = K^o - \mathcal{H}_n(rK_B^{v,1}, 0)G$. Как только он увидит, что $K_B'^s = K_B^{s,1}$, он будет знать, что транзакция адресована ему.[14]

   Для того чтобы найти выходы транзакции, предназначенные для его подадресов, Бобу нужен только его приватный ключ просмотра $k_B^v$ и публичный ключ траты подадреса $K_B^{s,1}$.

   src/crypto/crypto.cpp `derive_subaddress_public_key()`

4. Одноразовыми ключами для выхода будут:

$$K^o = \mathcal{H}_n(rK_B^{v,1}, 0)G + K_B^{s,1} = (\mathcal{H}_n(rK_B^{v,1}, 0) + k_B^{s,1})G$$
$$k^o = \mathcal{H}_n(rK_B^{v,1}, 0) + k_B^{s,1}$$

Теперь публичный ключ транзакции Элис предназначен для Боба ($rK_B^{s,1}$ вместо $rG$). Если Элис создаст транзакцию с $p$ выходов, где по крайней мере один выход будет предназначен для подадреса, ей потребуется создать уникальный публичный ключ транзакции для каждого выхода $t \in \{0, ..., p-1\}$. Другими словами, если Элис решит отправить средства на подадрес Боба ($K_B^{v,1}, K_B^{s,1}$) и на адрес Кэрол ($K_C^v, K_C^s$), ей придётся включить в данные транзакции два публичных ключа транзакции $\{r_1 K_B^{s,1}, r_2 G\}$.[15,16]

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_tx_with_tx_key()`

[14] Продвинутый злоумышленник может преуспеть в связывании подадресов [56] (атака, известная под названием «атака Януса»). При наличии подадресов $K_B^1$ $K_B^2$ (один из которых может быть нормальным адресом) злоумышленник может счесть, что они связаны. В этом случае он создаст выход транзакции $K^o = \mathcal{H}_n(rK_B^{v,2}, 0)G + K_B^{s,1}$ и включит в неё публичный ключ $rK_B^{s,2}$. Боб вычислит $rK_B^{v,2}$, чтобы найти $K_B'^{s,1}$, но он при этом не будет знать, что использовался ключ *другого* (под)адреса! Если он сообщит злоумышленнику, что получил средства на $K_B^1$, то злоумышленник будет знать, что $K_B^2$ является связанным подадресом (или нормальным адресом). Поскольку использование подадресов находится вне области применения, как избежать подобной проблемы решают разработчики кошельков. Пока нам неизвестны случаи, когда это было сделано, и единственным вариантом решения проблемы является разработка кошельков, соответствующих требованиям. Потенциальные способы включают в себя сокрытие от злоумышленников информации о получении средств, включение зашифрованного приватного ключа транзакции $r$ в данные транзакции, использование подписи Шнорра с общим секретом, использующим $K^{s,1}$ в качестве базовой точки вместо $G$, а также включение $rG$ в данные транзакции и верификацию общего секрета при помощи $rK^{s,1} \stackrel{?}{=} (k^s + \mathcal{H}_n(k^v, 1)) * rG$ (требует приватного ключа траты). Выходы, полученные на нормальный адрес, также должны верифицироваться. См. работу [101], в которой обсуждаются самые последние способы решения данной проблемы.

[15] 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.

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_account_address_as_str()`

[16] При использовании дополнительных публичных ключей транзакции возникает некоторая путаница (см. переход кода `transfer_selected_rct()` → `construct_tx_and_get_tx_key()` → `construct_tx_with_tx_key()` → `generate_output_ephemeral_keys()` и `classify_addresses()`) , связанная с выходами сдачи, если автору транзакции известен ключ просмотра получателя (так как это он сам; ещё один пример фиктивных выходов сдачи, созданных при необходимости в выходах с нулевой суммой, так как авторы генерируют эти адреса). Всякий раз, когда имеется по крайней мере два выхода, не относящихся к сдаче, и по крайней мере один из них принимается на подадрес, всё происходит нормальным образом, как говорилось выше (существующий на данный момент баг в основной реализации добавляет дополнительный публичный ключ транзакции к данным транзакции даже сверх других дополнительных ключей, и он нигде не используется). Если на подадрес направляется либо только выход сдачи, либо имеется только один выход, не относящийся к сдаче, и он направляется на подадрес, создаётся только один публичный ключ транзакции. В первом случае публичным ключом транзакции является $rG$ , а одноразовым ключом сдачи (с индексом подадреса 1, использующим нижний индекс $c$ для обозначения ключей сдачи) является $K^o = \mathcal{H}_n(k_c^v rG, t)G + K_c^{s,1}$, использующий нормальный ключ просмотра и ключ траты подадреса. В последнем случае публичный

## 4.4 Интегрированные адреса

Для того чтобы отличить между собой полученные выходы, получатель может попросить у отправителей включить в данные транзакции *идентификатор (ID) платежа*.[17] 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 с точки зрения приватности получателей, которые могут случайно раскрыть свои действия.[18] 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.[19]

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_account_integrated_address_as_str()`

Отправители, которые направляют выходы на интегрированные адреса, могут зашифровать ID платежа, используя общий секрет $rK_t^v$ и операцию XOR (см. подпункт 2.5), чтобы получатель мог потом расшифровать идентификатор при помощи соответствующего публичного ключа транзакции или другой процедуры XOR [9]. Шифрование ID платежа подобным образом позволяет отправителям доказать, что они провели определённую транзакцию (то есть в случае аудита, возмещения средств и других подобных случаях).[20]

### Шифрование

Отправитель зашифровывает каждый ID платежа для его включения в данные транзакции[21]

src/device/device_default.cpp `encrypt_payment_id()`

ключ транзакции $rK^{v,1}$ основан на ключе просмотра подадреса, а одноразовым ключом сдачи является $K^o = \mathcal{H}_n(k_c^v * rK^{v,1}, t)G + K_c^s$. . Эти подробности позволяют смешивать часть более часто используемых транзакций с нормальными адресами, транзакции с двумя выходами, которые на момент написания данного документа составляют примерно 95% от общего объёма транзакций.

[17] В настоящее время Monero поддерживает только один ID платежа на транзакцию, независимо от того, сколько выходов в ней используется.

[18] Эти длинные (256 бит) ID платежа в форме простого текста были исключены после выхода версии v0.15 основного программного обеспечения (одновременно с выходом версии v12 протокола в ноябре 2019). Несмотря на то, что некоторые кошельки могут поддерживать их и включать в данные транзакций по сей день, основной кошелёк будет игнорировать такие идентификаторы.

[19] Насколько известно автору, интегрированные адреса были реализованы только для нормальных адресов.

[20] Поскольку наблюдатель может распознать различие между транзакциями с ID платежей и без них, их использование делает историю транзакций менее единообразной. Поэтому, начиная с версии v10 протокола, основной вариант реализации предусматривает добавление фиктивных зашифрованных ID платежа во все транзакции с двумя выходами. Расшифровка одной вскроет все нулевые (это не является требованием протокола, но представляет собой хороший практический способ).

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_tx_with_tx_key`

[21] В случае Monero ID платежа в интегрированных адресах традиционно имеют длину 64 бита.

$$k_{\text{mask}} = \mathcal{H}_n(rK_t^v, \text{pid\_tag})$$
$$k_{\text{payment ID}} = k_{\text{mask}} \rightarrow \text{сокращение до длины ID платежа в битах}$$
$$\text{зашифрованный ID платежа} = \text{XOR}(k_{\text{payment ID}}, \text{ID платежа})$$

Мы используем pid_tag, чтобы $k_{\text{mask}}$ гарантированно отличался от компонента $\mathcal{H}_n(rK_t^v, t)$ в одноразовых адресах выходов.[22]

### Расшифровка

Любой получатель, для которого был создан ID платежа, может найти его, используя свой ключ просмотра и публичный ключ транзакции $rG$:[23]

src/device/device.hpp `decrypt_payment_id()`

$$k_{\text{mask}} = \mathcal{H}_n(k_t^v rG, \text{pid\_tag})$$
$$k_{\text{payment ID}} = k_{\text{mask}} \rightarrow \text{сокращение до длины ID платежа в битах}$$
$$\text{ID платежа} = \text{XOR}(k_{\text{payment ID}}, \text{зашифрованный ID платежа})$$

Подобным образом отправители могу расшифровывать ID платежа, который они ранее зашифровали, путём пересчёта общего секрета.

## 4.5 Адреса, использующие мультиподписи

Иногда бывает полезно разделить владение средствами между различными людьми/адресами. Этой теме посвящена Глава 9.

[22] В случае Monero pid_tag = `ENCRYPTED_PAYMENT_ID_TAIL` = 141. В транзакциях с множеством входов, например, мы вычисляем $\mathcal{H}_n(rK_t^v, t) \pmod l$, чтобы гарантировано использовать скалярную величину, которая будет меньше порядка подгруппы EC, но так как $l$ составляет 253 бита, а идентификаторы платежей всего 64 бита, использование модуля для шифровки ID платежей не будет иметь смысла, поэтому мы этого и не делаем.

[23] В данных транзакции не указывается, к какому выходы «принадлежит» ID платежа. Получателям необходимо указывать собственные ID платежей.

# ГЛАВА 5

# Сокрытие суммы Monero

В случае с большинством криптовалют, таких как Bitcoin, в примечаниях выходов транзакций, дающих право на то, чтобы потратить какую-либо «сумму» денег, такая сумма указывается простым текстом. Это позволяет наблюдателям без труда проверить соответствие суммы, которая тратится, отправляемой сумме.

В Monero используются *обязательства*, которые скрывают суммы в выходах от всех остальных, кроме отправителя и получателя. При этом наблюдатели сохраняют уверенность в том, что сумма транзакции более или менее равна тому, что действительно тратится. Как можно будет увидеть, обязательства по сумме также должны иметь соответствующие «доказательства диапазона», которые доказывают, что скрытая сумма находится в пределах некоего легитимного диапазона.

## 5.1 Обязательства

В общих чертах криптографическая *схема обязательства* является способом предоставления доказательства суммы без раскрытия самой суммы. После того как вы дадите обязательство, вам будет необходимо придерживаться его.

Например, в игре с подбрасыванием монеты Элис может конфиденциально дать обязательство по одному результату (то есть «назвать его»), опубликовав значение, хешированное с использованием секретных данных, по которому она даёт обязательство. После того как Боб 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 $blah$, а значение, по которому она хочет дать обязательство, — $heads$. Она хеширует $h = \mathcal{H}(blah, heads)$ и передаёт Бобу $h$. Боб подбрасывает монету, после чего Элис сообщает ему секретный ряд $blah$ и говорит, что дала обязательство по $heads$. После этого Боб вычисляет $h' = \mathcal{H}(blah, heads)$. И если $h' = h$, то он знает, что Элис указала $heads$ до броска монеты.

Элис использует так называемую «соль», $blah$, поэтому Боб не может просто угадать $\mathcal{H}(heads)$ и $\mathcal{H}(tails)$ до того, как подбросит монету, и выяснить, что она дала обязательство по $heads$.[1]

## 5.2 Обязательства Педерсена

*Обязательства Педерсена* [113] обладают свойством *аддитивной гомоморфности*. Если $C(a)$ и $C(b)$ являются обозначением обязательств по $a$ и $b$, соответственно, то $C(a + b) = C(a) + C(b)$.[2] 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 правило сохраняется:

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

Очевидно, определяя обязательство просто как $C(a) = aG$, мы бы смогли создать обманные таблицы обязательств, которые помогли бы нам распознать общие значения $a$.

Для обеспечения информационно-теоретической приватности необходимо ввести секретный *скрывающий фактор* и ещё один генератор $H$, которые не позволят узнать, для какого значения $\gamma$ будет действительным следующее равенство: $H = \gamma G$. Сложность решения задачи дискретного логарифма гарантирует, что вычисление $\gamma$ на основе $H$ просто невозможно.[3]

Следовательно, мы можем определить обязательство по $a$ как $C(x, a) = xG + aH$, где $x$ является скрывающим фактором (также известным как «маска»), который не даёт наблюдателям раскрыть $a$.

Обязательство $C(x, a)$ является информационно-теоретически приватным, так как существует

[1] Если значение, по которому было дано обязательство, очень трудно угадать и проверить, например, если оно является случайной точкой на эллиптической кривой, то в использовании «соли» в обязательстве нет никакой необходимости.

[2] Аддитивная гомоморфность в данном случае означает сохранение добавления при преобразовании скалярных величин в точки EC, применяя (для скалярной величины $x$, $x \to xG$.

[3] В случае Monero, $H = 8 * to\_point(\mathcal{H}_n(G))$. Это отличается от хеш-функции $\mathcal{H}_p$ тем, что выход $\mathcal{H}_n(G)$ интерпретируется напрямую как координата сжатой точки, а не производится математического вычисления точки на кривой (см. [107]). Исторические причины такого расхождения нам не известны, и, по сути, это единственный случай, когда $\mathcal{H}_p$ не используется (Bulletproofs также используют $\mathcal{H}_p$). Следует отметить наличие операции $*8$, которая гарантирует, что полученная точка будет находиться в нашей подгруппе $l$ ($\mathcal{H}_p$ также гарантирует это).

src/ringct/rctTypes.h
tests/unit_tests/ringct.cpp `TEST(ringct, HPow2)`

множество возможных комбинаций $x$ и $a$, которые будут иметь своим результатом $C$.[4] Если значение $x$ действительно будет случайным, у злоумышленника буквально не будет способа найти $a$ [88, 122].

## 5.3 Обязательства по сумме

В случае Monero суммы, содержащиеся в выходах, сохраняются в транзакциях в виде обязательств Педерсена. Мы определяем обязательство по сумме выхода $b$ следующим образом:

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

src/ringct/rctOps.cpp `addKeys2()`

У получателей должна быть возможность узнавать, сколько денег находится в каждом выходе, имеющемся у них, а также они должны быть способны восстановить обязательство, чтобы они могли использовать эти выходы в качестве входов в новых транзакциях. Это означает, что скрывающий фактор $y$ и сумма $b$ должны быть сообщены получателю.

Нами применяется решение в виде общего секрета Диффи-Хеллмана $rK_B^v$, использующего «публичный ключ транзакции» (см. подпункт 4.2.1). В случае с каждой отдельно взятой транзакцией в блокчейне у каждого из её выходов $t \in \{0, ..., p-1\}$ имеется маска $y_t$ которую могут приватно вычислить как отправители, так и получатели, а также *сумма*, которая сохраняется в данных транзакции. В то время как $y_t$ является скалярной величиной эллиптической кривой и занимает 32 байта, размер $b$ будет ограничен 8 байтами при помощи доказательства диапазона, поэтому требуется сохранить значение, составляющее 8 байтов.[5,6]

$$y_t = \mathcal{H}_n(\text{“}commitment\_mask\text{”}, \mathcal{H}_n(rK_B^v, t))$$
$$amount_t = b_t \oplus_8 \mathcal{H}_n(\text{“}amount\text{]}, \mathcal{H}_n(rK_B^v, t))$$

src/ringct/rctOps.cpp `ecdh-Encode()`

В данном случае $\oplus_8$ означает операцию XOR (см. подпункт 2.5), выполняемую между первыми 8 байтами операнда ($b_t$, размер которого уже составляет 8 байт, и $\mathcal{H}_n(...)$размер которого равен 32 байтам). Получатели могут применить ту же операцию XOR к $amount_t$, чтобы раскрыть $b_t$.

Получатель Боб сможет вычислить скрывающий фактор $y_t$ и сумму $b_t$, воспользовавшись публичным ключом транзакции $rG$ и своим ключом просмотра $k_B^v$, также он может проверить соответствие обязательства $C(y_t, b_t)$, полученного с данными транзакции и обозначенного как

[4] 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. Это свойство также известно как «идеальное сокрытие» [138]. Кроме того, даже лицо, публикующее обязательства, не сможет найти другой комбинации, не решив DLP для $\gamma$. Это свойство известно как «вычислительное связывание» [138].

[5] Как с одноразовым адресом $K^o$, о котором говорилось в подпункте 4.2, перед хешированием к общему секрету прикрепляется индекс выхода $t$. Это гарантирует, что выходы, направляемые по одному адресу, не будут иметь одинаковой маски и *суммы*, за исключением ничтожной вероятности. Так же как и ранее, член $rK_B^v$ умножается на 8, поэтому, по сути, является $8rK_B^v$.

[6] Это решение (реализованное в версии v10 протокола) заменило собой метод, который использовался ранее и требовал большего объёма данных, в результате чего тип транзакций изменился с версии v3 (`RCTTypeBulletproof`) на версию v4 (`RCTTypeBulletproof2`). Метод, который использовался раньше, рассматривался в предыдущей редакции настоящего отчёта [33].

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_tx_with_tx_key()` и в случае `generate_output_ephemeral_keys()`

$C_t^b$, имеющейся у него сумме.

В более общем плане любая третья сторона, обладающая доступом к ключу просмотра Боба, может расшифровать суммы в его выходах и убедиться в их соответствии связанным с ними обязательствам.

## 5.4 Введение в RingCT

Транзакция должна содержать ссылки на выходы других транзакций (которые скажут наблюдателям, какие из старых выходов можно потратить). Содержимое выхода включает в себя одноразовый адрес (указывающий на принадлежность выхода) и обязательство по выходу, скрывающее сумму (а также зашифрованную сумму выхода, о которой говорилось в подпункте 5.3).

Несмотря на то, что верификаторам транзакций не известно, сколько денег содержится в каждом входе и каждом выходе, им всё же необходимо знать, что сумма во входах равна сумме в выходах. Для этого Monero использует технологию под названием RingCT [108], которая была впервые внедрена в январе 2017 (версия v4 протокола).

Если у нас есть транзакция с $m$ выходов, содержащих $a_1, ..., a_m$ и $p$ выходов, содержащих $b_0, ..., b_{p-1}$, наблюдатель обоснованно может предположить, что:[7]

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

Поскольку обязательства являются аддитивными и нам не известно значение $\gamma$, мы можем с лёгкостью доказать наблюдателям, что наши входы равны выходам, сделав сумму обязательств по входам и выходам равным нулю (то есть задав сумму скрывающих факторов выходов как равную сумме скрывающих факторов старых входов):[8]

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

Чтобы избежать идентификации отправителя, мы используем несколько иной подход. Суммы, которые тратятся, соответствуют выходам предыдущих транзакций, по которым имеются обязательства

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

Отправитель может создать новые обязательства по тем же суммам, но используя разницу между двумя обязательствами:

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

[7] Если общая заданная сумма входов не равна любой комбинации имеющихся выходов, авторы транзакции могут добавить выход «сдачи», отправив дополнительные деньги самим себе. По аналогии с наличными деньгами, если у вас есть банкнота достоинством 20$ и вы тратите 15$, кассир вернёт вам 5$.

[8] Как указано в подпункте 2.3.1, мы можем выделить точку, инвертировав её координаты, а не добавляя её. Если $P = (x, y)$, $-P = (-x, y)$. Также можно вспомнить, что согласование элементов поля вычисляется $(\mathrm{mod}\ q)$, поэтому $(\breve{}x\ (\mathrm{mod}\ q))$.

Очевидно, что отправителю будет известен приватный ключ к разнице между двумя обязательствами:

$$C_j^a - C_j'^a = (x_j - x'_j)G$$

Следовательно, отправитель сможет использовать это значение как *обязательство по нулю*, так как он способен создать подпись с приватным ключом $(x_j - x'_j) = z_j$ и доказать отсутствие компонента $H$ в сумме (при этом подразумевается, что значение $\gamma$ не известно). Другими словами, доказать, что $C_j^a - C_j'^a = z_j G + 0H$, что будет нами сделано в Главе 6, когда мы будем рассматривать структуру транзакций RingCT.

Назовём $C_j'^a$ а *псевдообязательством по выходу*. Псевдообязательства по выходам включаются в данные транзакции, и берётся по одному такому обязательству для каждого входа.

Перед передачей транзакции в блокчейн сеть захочет верифицировать сбалансированность сумм. Скрывающие факторы псевдообязательств по выходам выбираются так, чтобы

$$\sum_j x'_j - \sum_t y_t = 0$$

Это позволяет нам доказать, что суммы в выходах равны суммам во входах:

$$(\sum_j C_j'^a - \sum_t C_t^b) = 0$$

src/ringct/rctSigs.cpp verRctSemanticsSimple()

К счастью, выбрать такие скрывающие факторы просто. В текущей версии Monero все скрывающие факторы являются случайными для псевдообязательства $m^{\text{th}}$, где $x'_m$ является просто

$$x'_m = \sum_t y_t - \sum_{j=1}^{m-1} x'_j$$

genRctSimple()

## 5.5 Доказательства диапазона

Проблема с аддитивными обязательствами состоит в том, что если у нас есть обязательства $C(a_1)$, $C(a_2)$, $C(b_1)$ и $C(b_2)$ и мы собираемся использовать их для доказательства того, что $(a_1 + a_2) - (b_1 + b_2) = 0$, то эти обязательства будут по-прежнему применимы, если одно из значений в уравнении будет «отрицательным».

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

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

где

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

Так как $-10 = l - 10$, мы, по сути, создали на $l$ больше Monero (более $7.2\text{x}10^{74}$!), чем вложили.

У этой проблемы Monero есть решение, которое заключается в доказательстве того, что сумма каждого выхода находится в определённом диапазоне (от 0 до $2^{64} - 1$), при помощи схемы

доказательства Bulletproofs, впервые описанной Бенедиктом Бюнцем *и др.* в работе [43] (а также рассматриваемой в работах [138, 50]).[9] Учитывая сложную и запутанную природу алгоритма Bulletproofs, мы не рассматриваем его в этом документе. Кроме того, мы считаем, что документы, на которые мы ссылаемся, в достаточной мере освещают соответствующие концепции.[10]

Алгоритм доказательства Bulletproof в качестве входа берёт суммы выходов $b_t$ и маски обязательств $y_t$ и выводит все $C_t^b$ , а также состоящее из $n$ элементов агрегированное доказательство $\Pi_{BP} = (A, S, T_1, T_2, \tau_x, \mu, \mathbb{L}, \mathbb{R}, a, b, t)$[11,12]. Это единственное доказательство используется для того, чтобы доказать, что все суммы выходов одновременно находятся в одном диапазоне, а их накопление значительно снижает требование к занимаемому месту (вместе с тем увеличивая время верификации).[13] В качестве входа алгоритм верификации берёт все $C_t^b$ и $\Pi_{BP}$ и выводит `true`, если все суммы, по которым были предоставлены обязательства, находятся в диапазоне от 0 до $2^{64} - 1$.

src/ringct/rctSigs.cpp `proveRangeBulletproof()`

$\Pi_{BP}$, состоящее из $n$ элементов, занимает $(2 \cdot \lceil \log_2(64 \cdot p) \rceil + 9) \cdot 32$ байт.

[9] Понятно, что при наличии нескольких выходов в легитимном диапазоне сумма содержащихся в них сумм сможет превысить допустимое значение, что вызовет появление проблемы. Тем не менее, если максимальный выход гораздо меньше, чем $l$, для того чтобы проблема возникла, требуется просто огромное количество выходов. Например, если допустимый диапазон составляет от 0 до 5, а $l = 99$, значит, чтобы подделать деньги, используя 2 выхода, нам понадобится $5 + 5 + \ldots + 5 + 1 = 101 \equiv 2 \pmod{99}$ для 21 выхода. В Monero $l$ примерно в 2^189 больше доступного диапазона, а это означает, что для подделки денег потребуется невероятное количество выходов, равное 2^189.

[10] До перехода к версии v8 протокола доказательства диапазона дополнялись подписями Борромео, которые мы рассмотрели в первой редакции «От нуля к Monero» [33].

[11] Каждый из векторов $\mathbb{L}$ и $\mathbb{R}$ содержит $\lceil \log_2(64 \cdot p) \rceil$ элементов. $\lceil\ \rceil$ означает, что логарифмическая функция округляется. Из-за своей структуры некоторые Bulletproofs используют «фиктивные выходы» для заполнения, а также чтобы гарантировать, что $p$ плюс определённое количество фиктивных выходов будет иметь степень 2. Такие фиктивные выходы могут быть сгенерированы во время верификации, и они не сохраняются с данными доказательства.

[12] Переменные, используемые в доказательстве Bulletproof, никак не связаны с другими переменными, о которых говорится в данном документе. Любое совпадение символов является совершенно случайным. Следует отметить, что групповые элементы $A, S, T_1, T_2, \mathbb{L}$, и $\mathbb{R}$ перед сохранением умножаются на $1/8$, а затем, во время верификации, умножаются на 8. Это гарантирует, что все они будут входить в подгруппу $l$ (см. подпункт 2.3.1).

[13] Получается, что множество отдельных доказательств Bulletproofs может быть объединено в группы. А это значит, что их можно будет верифицировать одновременно. Это улучшает время их верификации, и сейчас доказательства Bulletproofs Monero объединяются в группу поблочно. При этом нет никакого теоретического ограничения в отношении количества доказательств, которые можно объединить вместе. Каждая транзакция может иметь только одно доказательство Bulletproof.

rct/ringct/bulletproofs.cc `bulletproof_VERIFY()`

# глава 6

# Кольцевые конфиденциальные транзакции (RingCT)

В Главах 4 и 5 нами были рассмотрены несколько аспектов транзакций Monero. До этого момента проведение простой транзакции с одним входом и одним выходом некоторым неизвестным отправителем какому-то неизвестному получателю происходило примерно так:

«В моей транзакции используется публичный ключ $rG$. Я потрачу старый выход $X$ (следует отметить, что в нём имеется скрытая сумма $A_X$, обязательство по которой содержится в $C_X$). Я дам обязательство по псевдовыходу $C'_X$. Я создам один выход $Y$, который можно будет потратить по одноразовому адресу $K^o_Y$. В нём будет скрытая сумма $A_Y$, обязательство по которой будет содержаться в $C_Y$, которая будет зашифрована для получателя и которая будет доказана в определённом диапазоне в стиле доказательств диапазона Bulletproofs. Следует отметить, что $C'_X - C_Y = 0$».

Но остаются некоторые вопросы. Действительно ли отправитель является обладателем $X$? Действительно ли обязательство по псевдовыходу $C'_X$ соответствует $C_X$ так, что $A_X = A'_X = A_Y$? Может, кто-то вмешался в транзакцию и направил выход тому получателю, которому отправитель транзакции и не собирался отправлять его?

Как было сказано в подпункте 4.2, мы можем доказать факт обладания выходом, подписав сообщение его одноразовым адресом (тот, у кого имеется ключ от адреса, владеет и выходом). Мы также можем доказать, что в нём содержится та же сумма, что и в обязательстве по псевдовыходу, доказав знание приватного ключа к нулевому обязательству ($C_X - C'_X = z_X G$). Более того, если такое сообщение является *всеми данными транзакции* (кроме самой подписи), то верификатор может быть уверен в том, что всё происходит именно так, как того

хотел отправитель (подпись работает только в случае с оригинальным сообщением). MLSAG-подписи позволяют нам делать всё это, скрывая при этом выход, который тратится на самом деле, среди других выходов, взятых из блокчейна. Таким образом, внешний наблюдатель никогда не сможет сказать с уверенностью, какой из выходов является действительным.

## 6.1 Типы транзакций

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

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_tx_with_tx_key()`

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 при создании всех новых транзакций Monero, поэтому мы не станем описывать унаследованные типы транзакций, хотя они до сих пор частично поддерживаются.[1] Вплоть до этого момента и в дальнейшем мы будем рассматривать тип транзакций, называемый `RCTTypeBulletproof2`.[2]

В подпункте 6.3 нами приводится краткое описание концепции данных транзакций.

## 6.2 Конфиденциальные транзакции `RCTTypeBulletproof2`

В настоящее время (версия v12 протокола) при проведении всех транзакций требуется использовать именно этот тип, где каждый выход подписывается отдельно. Реальный пример транзакции `RCTTypeBulletproof2` со всеми её компонентами приводится в Приложении A.

### 6.2.1 Обязательства по сумме и комиссии за проведение транзакций

Допустим, отправителем транзакции ранее были различные выходы с суммами $a_1, ..., a_m$ направленные на одноразовый адрес $K^o_{\pi,1}, ..., K^o_{\pi,m}$ с обязательствами по сумме $C^a_{\pi,1}, ..., C^a_{\pi,m}$.

Этому отправителю известны приватные ключи $k^o_{\pi,1}, ..., k^o_{\pi,m}$, соответствующие одноразовым адресам (см. подпункт 4.2). Отправителю также известны скрывающие факторы $x_j$, используемые в обязательствах $C^a_{\pi,j}$ (см. подпункт 5.3).

[1] Протокол RingCT был впервые реализован в январе 2017 (в версии v4 протокола). Он стал обязательным для использования при создании всех новых транзакций в сентябре 2017 (в версии v6 протокола) [16]. RingCT является второй версией протокола транзакций Monero.

[2] За время существования RingCT произошёл отказ от трёх типов транзакций: `RCTTypeFull`, `RCTTypeSimple` и `RCTTypeBulletproof`. Первые две использовались в первой итерации RingCT и рассматриваются в первой редакции данного отчёта [33]. Затем, после появления Bulletproofs (версия v8 протокола), произошёл отказ от `RCTTypeFull`, а тип `RCTTypeSimple` был обновлён до `RCTTypeBulletproof`. `RCTTypeBulletproof2` появился по причине небольшого усовершенствования шифрования масок и сумм в обязательствах по выходам (v10).

Как правило, выходы транзакций в целом *ниже*, чем входы транзакций, что обеспечивает возможность выплаты комиссий майнерам за включение ими транзакции в блокчейн.[3] Суммы комиссий за проведение транзакций $f$ хранятся в форме обычного текста в данных транзакций, передаваемых в сеть. Майнер для себя может создать дополнительный выход с комиссией (см. подпункт 7.3.6).

Транзакция состоит из входов $a_1, ..., a_m$ и выходов $b_0, ..., b_{p-1}$ в результате чего получаем $\sum_{j=1}^{m} a_j - \sum_{t=0}^{p-1} b_t - f = 0$.[4]

Отправитель вычисляет обязательство по псевдовыходу для сумм во входах, $C'^a_{\pi,1}, ..., C'^a_{\pi,m}$, и создаёт обязательства для заданных сумм выходов $b_0, ..., b_{p-1}$. Обозначим эти обязательства как $C^b_0, ..., C^b_{p-1}$.

Ему известны приватные ключи $z_1, ..., z_m$ к нулевым обязательствам $(C^a_{\pi,1} - C'^a_{\pi,1}), ..., (C^a_{\pi,m} - C'^a_{\pi,m})$.

Для верификаторов, которым необходимо подтвердить, что сумма сумм транзакции равна нулю, сумма комиссии должна быть преобразована в обязательство. Решение заключается в вычислении обязательства по комиссии $f$ без маскирующего эффекта какого-либо «ослепляющего» фактора. То есть $C(f) = fH$.

Теперь мы можем доказать, что сумма входов равна сумме выходов:

src/ringct/rctSigs.cpp verRctSemanticsSimple()

$$(\sum_j C'^a_j - \sum_t C^b_t) - fH = 0$$

### 6.2.2 Подпись

Отправитель выбирает из блокчейна $m$ наборов дополнительных несвязанных адресов и их обязательства с размером $v$, соответствующих очевидно неотправленным выходам.[5,6] Чтобы

[3] Протокол Monero предполагает наличие минимального размера базовой комиссии, который масштабируется по мере увеличения веса транзакции. Это «полуобязательное» требование, поскольку, несмотря на то, что вы можете создавать новые блоки, содержащие транзакции с небольшой комиссией, большинство узлов Monero не станут ретранслировать такие транзакции другим узлам. В результате лишь немногие майнеры станут включать их в блоки (если вообще станут). Авторы транзакций могут по желанию увеличить размер комиссии сверх минимального. Более подробно эта проблема рассматривается в подпункте 7.3.4.

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_tx_with_tx_key()`

[4] В соответствии с базовым вариантом реализации выходы перемешиваются случайным образом до того, как им будет присвоен индекс, поэтому внешний наблюдатель не в состоянии произвести эвристический анализ их порядка. Входы сортируются образом ключа в рамках данных транзакции.

[5] В случае с Monero стандартный выбор набора «дополнительных несвязанных адресов», как правило, происходит в соответствии с определением гамма-распределения по всему ряду старых выходов (только выходов RingCT — треугольное распределение применяется только в отношении выходов, которые появились до реализации RingCTRi). Данный метод подразумевает применение процедуры биннинга, чтобы сгладить различие в плотности блоков. Сначала вычисляется среднее время между созданием выходов транзакций, вплоть до года в случае с выходами RingCT (среднее время = [#кол-во выходов/#кол-во блоков]*время создания блока). При помощи гамма-распределения выбирается блок, а затем из блока выбирается случайный выход, который войдёт в набор. [93]

src/wallet/wallet2.cpp `get_outs()`

`gamma_picker::pick()`

[6] В версии v12 протокола все входы должны быть созданы по крайней мере 10 блоков назад (`CRYPTONOTE_DEFAULT_TX _SPENDABLE_AGE`). До выхода версии v12 в основном варианте реализации по

подписать вход $j$, он перемешивает набор с размером $v$, создавая *кольцо* с собственным непотраченным одноразовым адресом $j^{\text{th}}$ (с назначенным уникальным индексом $\pi$) и ложными нулевыми обязательствами следующим образом:[7]

$$\begin{aligned}\mathcal{R}_j = \{&\{K^o_{1,j}, (C_{1,j} - C'^a_{\pi,j})\},\\ &\dots\\ &\{K^o_{\pi,j}, (C^a_{\pi,j} - C'^a_{\pi,j})\},\\ &\dots\\ &\{K^o_{v+1,j}, (C_{v+1,j} - C'^a_{\pi,j})\}\}\end{aligned}$$

Чтобы подписать своё кольцо, где ей известны приватные ключи $k^o_{\pi,j}$ для $K^o_{\pi,j}$, Элис использует MLSAG-подпись (см. подпункт 3.5) и $z_j$ в качестве нулевого обязательства ($C^a_{\pi,j}$ - $C'^a_{\pi,j}$). Поскольку в случае с нулевым обязательством в образе ключа нет необходимости, то и в структуре подписи такого компонента тоже не будет.[8]

src/ringct/rctSigs.cpp proveRctMGSimple()

Каждый вход транзакции подписывается индивидуально при помощи таких колец, как $\mathcal{R}_j$, о чём говорилось выше. Это позволяет скрыть выходы, которые тратятся на самом деле, $(K^o_{\pi,1}, ..., K^o_{\pi,m})$, среди других непотраченных выходов.[9] Поскольку определённая часть каждого кольца содержит нулевое обязательство, используемое обязательство по псевдовыходу должно содержать сумму, равную сумме входа, который реально тратится. Это связывает входы с доказательством равенства сумм. При этом не раскрывается, какой из участников кольца является реальным входом.

Сообщение $\mathfrak{m}$, подписываемое каждым входом, по сути, является хешем всех данных транзакций, *за исключением* MLSAG-подписей.[10] Это гарантирует защиту транзакций от внесения

умолчанию использовалось 10 блоков, но это не было обязательным требованием, поэтому любой кошелёк мог использовать иное количество блоков, и в некоторых случаях это было действительно так [82].

[7] В случае с Monero каждое кольцо любой транзакции должно быть одинакового размера, а протокол регулирует количество участников кольца на каждый выход, который тратится. Это количество менялось от версии к версии: v2 (март 2016) $\geq$ 3, v6 (сентябрь 2017) $\geq$ 5, v7 (апрель 2018) $\geq$ 7, 7, v8 (октябрь 2018) — только 11. Начиная с версии v6 ни одно из колец не могло содержать повторяющихся участников. Тем не менее в разных кольцах участники могли повторяться, что позволяло использовать множество входов при недостаточном количестве выходов в истории транзакций (то есть можно было создавать кольца без пересечения) [135].

src/cryptonote_core/cryptonote_core.cpp check_tx_inputs_ring_members_diff()

[8] При построении и верификации подписи исключается член $r_{i,2}\mathcal{H}_p(C_{i,j} - C'^a_{\pi,j}) + c_i\tilde{K}_{z_j}$.

[9] Преимущество раздельного подписания входов состоит в отсутствии необходимости расположения реальных входов и нулевых обязательств по одному и тому же индексу $\pi$, как это было бы в случае агрегации. Это означает, что даже если один из источников входа станет известен, остальные останутся скрытыми. Старый тип транзакций `RCTTypeFull` подразумевал использование агрегированных подписей, где все кольца объединялись в одно, и именно по этой причине мы отказались от него.

[10] Фактическим сообщением является $\mathfrak{m} = \mathcal{H}(\mathcal{H}(tx\_prefix), \mathcal{H}(ss), \mathcal{H}(\text{range proofs}))$, где:

src/ringct/rctSigs.cpp get_pre_mlsag_hash()

$tx\_prefix$ ={версия эры транзакции (то есть RingCT = 2), входы {офсеты ключей участников кольца, образы ключей}, выходы {одноразовые адреса}, дополнительные данные {публичный ключ транзакции, ID платежа или прочее,}}

$ss$ ={тип транзакции (`RCTTypeBulletproof2` = '4'), комиссия за проведение транзакции, обязательства по псевдовыходам для соответствующих входов, ecdhInfo (зашифрованные суммы), обязательства по выходам}.

Терминология определяется в Приложении A.

изменений как с точки зрения её авторов, так и верификаторов. Создаётся только одно сообщение, и оно подписывается каждым входом MLSAG.

Одноразовый приватный ключ $k^o$ является сущностью модели транзакций Monero. Подписание сообщения $\mathfrak{m}$ при помощи $k^o$ доказывает, что вы владеете суммой, обязательство по которой даётся в $C^a$. Верификаторы могут быть уверены в том, что авторы транзакции тратят собственные средства, даже не зная, какие средства тратятся, сколько тратится и сколько ещё средств остаётся у таких авторов!

### 6.2.3 Исключение возможности двойной траты

Подпись MLSAG (см. подпункт 3.5) содержит образы $\tilde{K}_j$ 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.

src/cryptonote_core/blockchain.cpp `have_tx_keyimges_as_spent()`

Сети необходимо только проверить, чтобы образы ключей, включённые в MLSAG-подписи (соответствующие входам и вычисленные как $\tilde{K}^o_j = k^o_{\pi,j}\mathcal{H}_p(K^o_{\pi,j})$), не встречались ранее в других транзакциях.[11] Если встречались, то вы можете быть уверенными в том, что столкнулись с попыткой двойной траты выхода $(C^a_{\pi,j}, K^o_{\pi,j})$.

### 6.2.4 Требования к занимаемому месту

#### Подпись MLSAG (входы)

Как мы помним из подпункта 3.5, MLSAG- подпись в данном контексте будет выражена как

$$\sigma_j(\mathfrak{m}) = (c_1, r_{1,1}, r_{1,2}, ..., r_{v+1,1}, r_{v+1,2}) \text{ with } \tilde{K}^o_j$$

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

Учитывая всё это, а также сжатие точек (см. подпункт 2.4.2), поскольку каждое кольцо $\mathcal{R}_j$ содержит $(v+1)\cdot 2$ ключа, хранение подписи входа $\sigma_j$ потребует $(2(v+1)+1)\cdot 32$ байта. Кроме того, образ ключа $\tilde{K}^o_{\pi,j}$ и обязательство по псевдовыходу $C'^a_{\pi,j}$ занимают $(2(v+1)+3)\cdot 32$ байта на вход.

К этому значению добавим место, необходимое для хранения смещённых членов кольца в блокчейне (см. Приложение A). Эти офсеты (смещения) используются верификаторами для

[11] Верификаторы также должны проверить, является ли образ изображения членом подгруппы генератора (см. подпункт 3.4).

вычисления ключа и обязательств каждого выхода члена кольца MLSAG-подписи в блокчейне и хранятся как целые числа переменной длины, поэтому мы не можем точно количественно определить необходимое пространство.[12,13,14]

src/ringct/rctSigs.cpp verRctMGSimple() verRctSemanticsSimple()

Верификация всех MLSAG-транзакций типа `RCTTypeBulletproof2` предполагает вычисление $(C_{i,j} - C'^{a}_{\pi,j})$ и $(\sum_j C'^{a}_j \stackrel{?}{=} \sum_t C^b_t + fH)$, а также верификацию образов ключей в подгруппе $G$ при помощи $l\tilde{K} \stackrel{?}{=} 0$.

### Доказательства диапазона (выходы)

Совокупное доказательство Bulletproof требует $(2 \cdot \lceil \log_2(64 \cdot p) \rceil + 9) \cdot 32$ байт места.

src/ringct/bulletproofs.cpp `bulletproof_VERIFY()`

[12] См. работу [66] или [24] где рассматривается тип данных, называемый varint и используемый Monero. It is an integer type that uses up to 9 bytes, and stores up to 63 bits of information.

src/common/varint.h

[13] Представьте, что блокчейн содержит длинный список выходов транзакций. Мы указываем индексы выходов, которые хотим использовать в кольцах. Большие индексы требуют больше места для хранения. Нам требуется «абсолютное» положение *одного* индекса из каждого кольца и «относительное» положение индексов других участников кольца. Например, при наличии реальных индексов {7,11,15,20} нам необходимо указать {7,4,4,5}. Верификаторы могут вычислить последний индекс как (7+4+4+5 = 20). Участники внутри колец расположены в возрастающем порядке по индексам, имеющимся в блокчейне.

src/cryptonote_basic/cryptonote_format_utils.cpp `absolute_output_offsets_to_relative()`

[14] В случае с транзакцией с 10 входами, использующими кольца, состоящие из 11 участников, потребуется $((11 \cdot 2 + 3) \cdot 32) \cdot 10 = 8000$ байт для входов и примерно от 110 до 330 байт для офсетов (при наличии 110 участников колец).

## 6.3 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 Приложении A.

- Тип: '0' обозначает `RCTTypeNull` (для майнеров), а '4' обозначает `RCTTypeBulletproof2`
- Входы: для каждого входа $j \in \{1, ..., m\}$, потраченного автором транзакции
  - **Смещённые члены кольца**: список «офсетов», указывающий, где верификатор может найти членов кольца $j$'s входа $i \in \{1, ..., v+1\}$ в блокчейне (включающем реальный вход)
  - **Подпись MLSAG**: члены $\sigma_j$: $c_1$, $r_{i,1}$ и $r_{i,2}$ для $i \in \{1, ..., v+1\}$
  - **Образ ключа**: образ ключа $\tilde{K}_j^{o,a}$ для входа $j$
  - **Обязательство по псевдовыходу**: $C_j'^a$ для входа $j$
- Выходы: для каждого выхода $t \in \{0, ..., p-1\}$, направленного на адрес или подадрес $(K_t^v, K_t^s)$.
  - **Одноразовый (скрытый) адрес**: $K_t^{o,b}$ для выхода $t$
  - **Обязательство по выходу**: $C_t^b$ для выхода $t$
  - **Зашифрованная сумма**: чтобы владельцы выходов могли вычислить $b_t$ для выхода $t$
    - *Сумма*: $b_t \oplus_8 \mathcal{H}_n(\text{“}amount\text{”}, \mathcal{H}_n(rK_B^v, t))$
  - **Доказательство диапазона**: совокупное доказательство Bulletproof для всех сумм в выходах $b_t$
    - *Доказательство*: $\Pi_{BP} = (A, S, T_1, T_2, \tau_x, \mu, \mathbb{L}, \mathbb{R}, a, b, t)$
- Комиссия за проведение транзакции: сообщается простым текстом как умноженная на $10^{12}$ (то есть атомные единицы, см. раздел 7.3.1), 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, $r_t K_t^{s,i}$ for each subaddress'd output $t$ and $r_t G$ для каждого выхода $t$, направленного на обычный адрес, и, возможно, ID платежа (один на транзакцию самое большее).[15]

Наш окончательный вариант примера транзакции с одним входом и одним выходом будет выглядеть следующим образом: «В моей транзакции используется публичный ключ $rG$.

[15] Информация, содержащаяся в данном поле, не верифицируется, несмотря на то, что *она* подписывается MLSAG входов, поэтому, внесение изменений невозможно (за исключением ничтожной вероятности). Поле не ограничено по содержащимся в нём данным, пока соблюдается максимальный размер транзакции. Более подробная информация содержится в работе [81].

Я потрачу один их выходов из набора $\mathbb{X}$ (следует отметить, что в нём имеется скрытая сумма $A_X$ обязательство по которой содержится в $C_X$). Я являюсь владельцем выхода, который трачу (я создал MLSAG-подпись по одноразовому адресу, содержащемуся в $\mathbb{X}$), и он не использовался ранее (его образ ключа $\tilde{K}$ ещё не появлялся в блокчейне). Я дам по нему псевдообязательство $C'_X$. Я создам один выход $Y$, который можно будет потратить по одноразовому адресу $K^o_Y$. В нём будет скрытая сумма $A_Y$, обязательство по которой будет содержаться в $C_Y$, которая будет зашифрована для получателя и которая будет доказана в определённом диапазоне в стиле доказательств диапазона Bulletproofs. Моя транзакция включает комиссию за транзакцию $f$. Следует отметить, что $C'_X - (C_Y + C_f) = 0$ и что я подписал нулевое обязательство $C'_X - C_X = zG$, что означает, что сумма во входе равна сумме в выходе ($A_X = A'_X = A_Y + f$)Все данные транзакции подписаны с использованием моей MLSAG, поэтому внешний наблюдатель может быть уверен, что они не были изменены каким-либо образом».

### 6.3.1 Требования к занимаемому месту

В случае с транзакцией типа `RCTTypeBulletproof2` для сохранения занимаемого места нам требуется $(2(v+1)+2) \cdot m \cdot 32$ байт, а для сохранения совокупного доказательства диапазона Bulletproof требуется $(2 \cdot \lceil \log_2(64 \cdot p) \rceil + 9) \cdot 32$ байт.[16]

Прочие требования:

- образы ключей входов: $m * 32$ байта;
- одноразовые адреса выходов: $p * 32$ байта;
- обязательства по выходам: $p * 32$ байта;
- зашифрованные суммы в выходах: $p * 8$ байт;
- публичный ключ транзакции: как правило, 32 байта, $p * 32$ байта при отправке, по крайней мере, на один подадрес;
- ID платежа: 8 байт для интегрированных адресов (не более одного на транзакцию);
- комиссия за проведение транзакции: сохраняется как целое число переменной длины, то есть $\leq 9$ байт;
- офсеты входов: сохраняются как целые числа переменной длины, то есть $\leq 9$ байт на офсет для участников кольца $m * (v+1)$;
- время снятия блокировки: сохраняется как целое число переменной длины, то есть $\leq 9$ байт;[17]
- «дополнительные» теги: любые данные в «дополнительном» поле (например, публичный ключ транзакции) начинаются с «тега» размером 1 байт, а в случае с некоторыми данными «длина» может составлять 1+ байт (более подробная информация содержится в Приложении A).

[16] Количество содержимого транзакции ограничено так называемым максимальным «весом транзакции». До реализации доказательств Bulletproofs в версии v8 протокола (и на самом еле, сейчас транзакции включают в себя всего два выхода) вес и размер транзакции, выраженный в байтах, были одинаковыми. Максимальный вес составляет (0.5*300kB - `CRYPTONOTE_COINBASE_BLOB_RESERVED_SIZE`), где зарезервированное для блоба место (600 байт) предназначено для включения майнинговых транзакций в блок. До реализации версии v8 множитель 0.5x не использовался, а член 300kB в предшествующих версиях протокола был меньше (20kB в v1, 60kB в v2, 300kB в v5). Мы более подробно рассматриваем эту тему в подпункте 7.3.2.

src/cryptonote_core/tx_pool.cpp `get_transaction_weight_limit()`

[17] Автор любой транзакции может заблокировать её выходы, и их нельзя будет потратить до достижения определённой высоты блока (или до достижения определённой временной метки UNIX, после которой выходы можно будет добавить в кольцо транзакции, включаемой в блок). Все выходы могут быть заблокированы только до одной и той же высоты блока. Непонятно, имеет ли это какой-то практический смысл для авторов транзакций (возможно, это полезно в случае со смарт-контрактами). В случае с майнинговыми транзакциями выходы в обязательном порядке блокируются на 60 блоков. После реализации версии v12 протокола выходы по умолчанию блокируются на 10 блоков, что эквивалентно обязательному минимальному времени блокировки на 10 блоков. Если транзакция публикуется в 10-м блоке при времени блокировке, равном 25, её выходы можно будет потратить в 25-м блоке или позднее. В случае с обычными транзакциями время блокировки, пожалуй, одна из наиболее редко используемых функций Monero.

src/cryptonote_core/blockchain.cpp `is_tx_spendtime_unlocked()`

# ГЛАВА 7

# Блокчейн Monero

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 [96], а в цифровом мире мы можем предложить свои возможности всему миру.

Среда обмена (деньги) 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 [96]. 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 самая распространённая 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. [97]

Чтобы нивелировать эти недостатки были разработаны децентрализованные цифровые валюты.[1]

[1] Эта глава содержит более подробную информацию по реализации, чем предыдущие, так как природа блокчейна сильно зависит от специфики его структуры.

## 7.1 Цифровая валюта

Разработка цифровой валюты — дело нетривиальное. Существует три типа: личная, централизованная и распределённая. Следует помнить о том, что цифровая валюта — это просто набор сообщений, а «суммы», записанные в сообщениях, рассматриваются в качестве денежных единиц.

В случае с **«email-моделью»** любой может создать монеты (например, написав сообщение «У меня есть 5 монет») и любой может отправлять свои монеты снова и снова любому человеку, у которого есть email адрес. Эмиссия не ограничена, и ничто не мешает многократно тратить одни и те же монеты (двойная трата).

В случае с **«игровой моделью»**, когда вся криптовалюта хранится/регистрируется в одной центральной базе данных, пользователи надеются на честность хранителя. Наблюдатели не могут проверить новых монет, а хранитель может изменить правила в любой момент, или же его могут цензурировать сторонние лица, наделённые соответствующими полномочиями.

### 7.1.1 Распределённая / совместно используемая версия событий

В случае с цифровыми «совместно используемыми» деньгами на множестве компьютеров хранится запись каждой валютной транзакции. Когда транзакция совершается на одном компьютере, она передаётся на другие компьютеры и принимается, если она соответствует предварительно определённым правилам.

Пользователи выгодно используют свои монеты только в том случае, если другие пользователи принимают их в обмен на что-либо, и пользователи принимают только те монеты, которые считают легитимными. Чтобы повысить ценность своих монет, пользователи, что совершенно естественно, склонны следовать одному общепринятому набору правил при отсутствии какого-либо централизованного контролирующего органа.[2]

**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**: Пользователь может потратить денежную единицу лишь один раз.

**Правило 4**: Только то лицо, у которого находится денежная единица, может потратить её.

**Правило 5**: Количество денег в выходе транзакции должно быть равно тому количеству денег, которое тратится.

**Правило 6**: Транзакции должны иметь правильный формат.

[2] В политологии это называется точкой Шеллинга [64], социальным минимумом или смарт-контрактом.

Правила 2-6 уже охвачены схемой транзакций, описанной в Главе 6 и обеспечивающей преимущества, связанные с взаимозаменяемостью и приватностью, обеспечиваемыми неопределённостью подписанта, а также с анонимностью получателя средств и невозможностью прочтения передаваемой суммы. О Правиле 1 мы ещё поговорим ниже в этой главе.[3] При проведении транзакций используется криптография. Поэтому мы называем их содержимое *криптовалютой*.

Если два компьютера получают данные двух легитимных 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», так как возможно наличие двух различных копий, следующих одним и тем же правилам.

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

### 7.1.2 Простой блокчейн

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

Допустим, валюта будет изначально создана заявлением о «генезисе» (т.е. создании): «Да будет SampleCoin!» Мы называем это сообщение «блоком», и хеш этого блока будет следующим:

$$BH_G = \mathcal{H}(\text{«Да будет SampleCoin!»})$$

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.

$$BH_1 = \mathcal{H}(BH_G, TH_1, TH_2, ...)$$
$$BH_2 = \mathcal{H}(BH_1, TH_3, TH_4, ...)$$

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 очень простой «блокчейн».[4]

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.

[3] В случае с товарными деньгами, такими как золото, эти правила соблюдаются в физической реальности.

[4] Технически блокчейн является «направленным ациклическим графом» (DAG), при этом блокчейны типа Bitcoin являются его одномерным вариантом. Графы DAG содержат конечное количество узлов и 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. [6]

Если различные блоки, ссылающиеся на один и тот же предыдущий блок, будут опубликованы в одно и то же время, сеть узлов снова разветвится (произойдёт форк), так как каждый блок получит один из новых блоков перед другим (для простоты представьте себе, что у около половины узлов имеются по обеим сторонам форка).

## 7.2 Сложность

Если узлы могут публиковать новые блоки всякий раз, когда им вздумается, то сеть может разбиться и разойтись на множество различных в равной степени законных цепочек. Скажем, сейчас уходит 30 секунд на то, чтобы каждый участник сети гарантированно получил новый блок. Что было бы, если бы блоки отправлялись каждые 31, 15, 10 секунд и так далее?

Мы можем контролировать скорость создания новых блоков в сети. Если время, необходимое для создания нового блока, будет значительно выше времени, за которое предыдущий блок достигнет каждого узла, то с сетью ничего подобного не произойдёт.

### 7.2.1 Майнинг блока

Выход криптографической хеш-функции распределяется равномерно и очевидно не зависит от входа. Это означает, что при наличии потенциального входа его хеш в равной степени вероятно станет каждым отдельно взятым возможным выходом. Кроме того, на вычисление каждого отдельного взятого хеша уходит определённое количество времени.

Представьте хеш-функцию $\mathcal{H}_i(x)$, выходы которой являются числами в пределах от 1 до 100: $\mathcal{H}_i(x) \in_R^D \{1, ..., 100\}$.[5] При некотором заданном $x$ функция $\mathcal{H}_i(x)$ выбирает то же самое «случайное» число в диапазоне $\{1, ..., 100\}$ всякий раз, когда вы вычисляете её. На вычисление $\mathcal{H}_i(x)$ уходит одна минута..

Допустим, у нас есть сообщение $\mathfrak{m}$, и нам необходимо вычислить такой «нонс» $n$ (некоторое целое число), чтобы выходы $\mathcal{H}_i(\mathfrak{m}, n)$ были числом меньшим или равным *заданному значению* $t = 5$ (то есть $\mathcal{H}_i(\mathfrak{m}, n) \in \{1, ..., 5\}$).

Так как только $1/20^{\text{th}}$ выходов $\mathcal{H}_i(x)$ будет соответствовать целевому значению, потребуется примерно 20 попыток вычислить рабочий $n$ (а следовательно, вычисления займут 20 минут).

Мы называем поиск подходящего нонса *майнингом*, а публикация сообщения с его нонсом является *доказательством работы*, так как это доказывает, что мы нашли подходящий нонс (даже если нам повезло и для этого понадобился всего один хеш, или же правильный нонс был опубликован просто вслепую), и это может проверить каждый, вычислив $\mathcal{H}_i(\mathfrak{m}, n)$.

Теперь, допустим, у нас есть хеш-функция для создания доказательства работы $\mathcal{H}_{PoW} \in_R^D \{0, ..., m\}$, где $m$ является максимально возможным выходом. При наличии сообщения $\mathfrak{m}$

[5] Мы используем $\in_R^D$ , чтобы показать, что выход является детерминировано случайным.

(блока информации), нонса $n$ для майнинга и целевого значения $t$ мы можем вычислить ожидаемое количество хешей (*сложность* $d$) следующим образом: $d = m/t$. Если $\mathcal{H}_{PoW}(\mathfrak{m}, n) * d \leq m$, то $\mathcal{H}_{PoW}(\mathfrak{m}, n) \leq t$ и $n$ принимается.[6]

src/cryptonote_basic/difficulty.cpp `check_hash()`

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.[7]

### 7.2.2 Скорость майнинга

Допустим, что все узлы одновременно занимаются майнингом, 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 $u \in \{1, ..., b\}$), 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.[8] 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: $totalDifficulty = \sum_u d_u$.

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:[9]

$$hashSpeed \approx totalDifficulty/totalTime$$

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. Примечание: мы округляем значения, поэтому сложность никогда не будет нулевой:

$$newDifficulty = hashSpeed * targetTime$$

[6] В случае Monero записывается/вычисляется только сложность, так как для вычисления $\mathcal{H}_{PoW}(\mathfrak{m}, n) * d \leq m$ $t$ не требуется.

[7] Майнинг и верификация ассиметричны, поскольку верификация доказательства работы занимает какое-то время (одно вычисление алгоритма доказательства работы), и при этом не важно, какова сложность.

[8] Временные метки определяются сразу после того, как майнер *начинает* майнинг нового блока, поэтому они могу немного отставать от фактического момента публикации. Майнинг следующего блока начинается сразу же. Таким образом, временная метка появляется *после* того, как блок обозначит, сколько времени было потрачено на его вычисление майнерами.

[9] 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 $K^o \in_{ER} \mathbb{Z}_l$ (где ER = effectively random, т.е. фактическую случайность), у майнеров всегда будут уникальные блоки, за исключением ничтожно малой вероятности, поэтому попытка нахождения одних и тех же нонсов не имеет значения.

Нет никакой гарантии, что майнинг следующего 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 скорости хеширования сети, а блоки будут вычисляться близко к *targetTime*.[10]

### 7.2.3 Консенсус: самая большая совокупная сложность

Теперь у нас есть механизм разрешения конфликтов между форками блокчейна.

По определению блокчейн с самой высокой совокупной сложностью (всех блоков блокчейна), а следовательно, на построение которого пришёлся наибольший объём работы, считается реальной, легитимной версией. Если блокчейн расходится, 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 легитимными, то они легко смогут 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.[11]

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 по-прежнему расти). Это будет очень трубно сделать, если вы не контролируете 50% хешрейта сети и не можете обогнать другие майнеры. [97]

### 7.2.4 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

[10] Если мы допустим постоянное постепенное повышение хешрейта сети, то, так как новые значения сложности будут зависеть от *последних* хешей (то есть предшествующих даже малейшему повышению хешрейта) то можно ожидать, что фактическое время вычисления блоков в среднем будет чуть меньше, чем *targetTime*. В результате этого график эмиссии (см. Раздел 7.3.1) может быть уравновешен штрафами, связанными с увеличением веса блоков, о чём будет говориться в Разделе 7.3.3.

[11] Разработчики Monero успешно изменяли протокол 11 раз, и практически все пользователи и майнеры следовали каждому форку: версия v1 18 апреля 2018 (генезис-версия) [127]; версия v2 в марте 2016; версия v3 в сентябре 2016; версия v4 в январе 2017; версия v5 в апреле 2017; версия v6 в сентябре 2017; версия v7 в апреле 2018; версии v8 и v9 в октябре 2018; версии v10 и v11 в марте 2019; версия v12 в ноябре 2019. В README, выложенном в главном git-репозитории, можно найти краткое описание изменений, внесённых в протокол в рамках каждой версии.

src/hardforks/hardforks.cpp `mainnet_hard_forks[]`

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 могут 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, а затем отбросив первые посторонние значения $o$, а после и 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.

Перед тем, как отбросить сторонние значения, мы отсортировали временные метки, но *только* временные метки. Сложность блоков осталась не отсортированной. Мы используем совокупное значение сложности для каждого блока, которое представляет собой значение сложности этого блока плюс значение сложности всех предыдущих блоков в цепочке.

src/cryptonote_core/blockchain.cpp `get_difficulty_for_next_block()`

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

$$totalTime = choppedSortedTimestamps[w] - choppedSortedTimestamps[1]$$
$$totalDifficulty = choppedCumulativeDifficulties[w] - choppedCumulativeDifficulties[1]$$

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).[12,13]

src/cryptonote_config.h

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

src/cryptonote_basic/difficulty.cpp `next_difficulty()`

**Правило 1**: Следует полностью игнорировать генезис-блок (блок 0, где $d = 1$). 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 итоговые значения.

**Правило 3**: После $w$ блоков следует отбросить высокие и низкие посторонние значения, сведя отброшенную сумму до $b$ блоков. Если сумма предшествующих блоков (минус $w$ будет нечётной, следует удалить ещё одно низкое, а не высокое постороннее значение.

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

[12] В марте 2016 (версия v2 протокола) целевое значение времени блоков увеличилось с 1 минуты до 2 минут [14]. Другие параметры сложности всегда оставались такими же.

[13] Алгоритм определения сложности Monero может показаться недостаточно оптимальным, если сравнивать с другими современными алгоритмами [144]. Однако, к счастью, он «довольно просто адаптируется к эгоистичному майнингу» [51], что является его важной особенностью.

### Доказательство работы Monero (PoW)

В различных версиях протокола Monero использовала несколько различных хеш-алгоритмов доказательства работы (с размером выходов 32 байта). Оригинальный протокол, известный как Cryptonight, был относительно неэффективен применительно к архитектурам GPU, FPGA и ASIC [124], если сравнивать его работу с работой стандартных хеш-функций, таких как SHA256. В апреле 2018 (версия v7 протокола) этот алгоритм был немного изменён, чтобы сделать невозможным майнинг при помощи микросхем ASIC, разработанных под Cryptonight [29], для чего понадобилось добавление новых блоков. Другая, немного отличающаяся от предшествующей, версия алгоритма, Cryptonight V2, была реализована в октябре 2018 (v8) [11], а версия Crytonight-R (также основанная на Cryptonight, но с более значительными изменениями, которые уже нельзя было назвать небольшими правками) стала использоваться с новыми блоками уже в марте 2019 (v10) [12]. Радикально новый алгоритм доказательства работы под названием RandomX [70] был разработан и стал обязательным для использования при вычислении новых блоков в ноябре 2019 (v12), чтобы обеспечить долгосрочную защиту от ASIC-майнинга [15].

src/cryptonote_basic/cryptonote_tx_utils.cpp `get_block_longhash()`

src/crypto/rx-slow-hash.c

## 7.3 Денежная масса

В случае с криптовалютами, в основе которых лежит блокчейн технология, существует два основных механизма создания денег.

Используя первый способ, создатели валюты могут просто создавать монеты и распространять их среди людей посредством генезис-сообщения. Часто такой способ называют «эйрдропом» (airdrop). Иногда создатели криптовалют создают для самих себя большую сумму денег путём так называемого «премайнинга» (pre-mine). [20]

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 пути. 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 до бесконечности.

Monero основана на валюте, известной как Bytecoin, у которой был большой премайнинг, за которым последовали вознаграждения за майнинг блоков [13]. 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.

### 7.3.1 Вознаграждение за майнинг блока

Майнеры, перед тем как начать майнинг нонса, создают «майнинговую транзакцию» (miner transaction) без входов и, по крайней мере, с одним выходом.[14] Общая сумма выходов равна вознаграждению за блок плюс комиссии за проведение транзакции 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.

src/cryptonote_core/blockchain.cpp `validate_miner_transaction()`

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 заниматься майнингом новых блоков? Возможно, только из каких-либо альтруистических побуждений или из простого любопытства. Тем не менее наличие нескольких майнеров позволяет злоумышленникам собрать >50% хешрейта сети и без труда переписать историю блокчейна.[15] Вот ещё одна из причин, по которой размер вознаграждения за вычисление блоков Monero не падает до нуля.

При наличии вознаграждений за майнинг блоков между майнерами 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 завладеть >50% хешрейта.

#### Сдвиг разрядов

Сдвиг разрядов используется для вычисления базового вознаграждения за майнинг блока (как будет указано в подпункте 7.3.3, 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 эта последняя часть отбрасывается, «сдвигается» в никуда, и в результате у нас остаётся [0011] = 3.[16]

[14] Майнинговая транзакция может содержать любое количество выходов, несмотря на то, что на данный момент базовый вариант реализации позволяет включать только один. Кроме того, в отличие от обычных транзакций какие-либо чёткие ограничения по весу такой транзакции отсутствуют. Они ограничены лишь функционально максимальным размером блоков.

[15] По мере того, как злоумышленнику удаётся заполучить большую часть хешрейта (свыше 50%), тем меньше у него уходит на то, чтобы переписать всё более и более старые блоки. При наличии блока, созданного $x$ дней назад, и при скорости хеширования $v$, а также честной скорости хеширования $v_h$ ($v > v_h$), такое переписывание займёт $y = x * (v_h/(v - v_h))$ дней.

[16] Поразрядное смещение на $n$ разрядов эквивалентно целочисленному делению на $2^n$.

**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).[17] В самом начале базовое вознаграждение за блок Monero составляло B = (L-M) >> 20. Если M = 0, то в десятичном формате мы получим следующее:

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

Эти числа являются «атомными единицами» — 1 атомную единицу Monero нельзя поделить. 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 тиккер» Monero).

$$\frac{L}{10^{12}} = 18,446,744.073709551615$$
$$B_0 = \frac{(L - 0) >> 20}{10^{12}} = 17.592186044415$$

И так мы получаем вознаграждение за самый первый блок, которое, к слову, ушло человеку под псевдонимом thankful_for_today (который и запустил проект Monero) в генезис-блоке Monero [127] и составило 17.6 Moneroj! Убедиться в этом вы можете, ознакомившись с Приложением C.[18]

По мере майнинга всё большего количества блоков значение M растёт, что снижает размер вознаграждения 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 уже две минуты [14]. В целях соблюдения и сохранения «графика эмиссии»[19], то есть скорости создания денег, размер вознаграждения за майнинг блока был удвоен. Это просто означало, что после изменения размер вознаграждения за майнинг новых блоков стал вычисляться как (L-M) >> 19 вместо >> 20. В настоящее время базовое вознаграждение за майнинг блока является следующим:

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

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_block_reward()`

### 7.3.2 Динамическое весовое значение блока

Было бы здорово, если бы каждая новая транзакция сразу же включалась в блок. Но что, если бы кто-то со злым умыслом захотел включить сразу множество транзакций? Блокчейн, в котором сохраняется каждая транзакция, быстро бы разросся до чудовищных размеров.

[17] Теперь, возможно, вы понимаете, почему доказательства диапазона (см. подпункт 5.5) ограничивают суммы транзакции до 64 бит.

[18] В блокчейне суммы Monero сохраняются в формате атомных единиц.

[19] Интересное сравнение графиков эмиссии Monero и Bitcoin приводится в работе [19].

Одним из способов, позволяющих избежать этого, является фиксированный размер блоков (выраженный в байтах), то есть количество транзакций, включаемых в блок, ограничено. А что, если объём честных транзакций вырастет? Автору каждой транзакции придётся добиваться предоставления ему места в новом блоке, предлагая более высокую комиссию майнерам. Майнеры сосредоточат своё внимание на транзакциях с наибольшей комиссией. По мере роста объёма транзакций, комиссии станут непозволительно высокими для транзакций с небольшими суммами (как в случае, когда Элис покупала яблоко у Боба). Только транзакции тех людей, кто предложит больше остальных, попадут в блокчейн.[20]

Monero старается избегать подобных крайностей (неограниченного и фиксированного размера), используя динамическое весовое значение блоков.

### Размер и вес

После реализации алгоритма Bulletproofs (v8) размер транзакций и блоков уже не соблюдается так строго. Теперь используется термин *вес транзакции (transaction weight)*. Вес майнинговой транзакции (см. подпункт 7.3.6) и обычной транзакции с двумя выходами равен их размеру, выраженному в байтах. Если обычная транзакция содержит более двух выходов, её вес будет несколько больше размера.

src/cryptonote_basic/cryptonote_format_utils.cpp `get_transaction_weight()`

Как было сказано в подпункте 5.5, доказательство Bulletproof занимает $(2 \cdot \lceil \log_2(64 \cdot p) \rceil + 9) \cdot 32$ байт. Поэтому по мере добавления выходов место, необходимое для сохранения доказательств диапазона, будет сублинейно возрастать. Тем не менее верификация Bulletproofs происходит линейно, поэтому искусственное повышение веса транзакций «стоит» дополнительного времени верификации (это называется «компенсацией затрат»).

Предположим, что у нас есть транзакция с $p$ выходов, и представим, что если $p$ не в степени 2, мы можем создать достаточное количество ложных выходов, чтобы заполнить пробел. Мы вычисляем разницу между фактическим размером доказательства Bulletproof и размером всех доказательств Bulletproof, если эти $p$ + «ложные выходы» были в транзакциях с двумя выходами (это будет $p = 2$). Мы компенсируем только 80% разницы.[21]

src/cryptonote_basic/cryptonote_format_utils.cpp `get_transaction_weight_clawback()`

$$\text{transaction\_clawback} = 0.8 * [(23 * (p + \text{num\_dummy\_outs})/2) \cdot 32 - (2 \cdot \lceil \log_2(64 \cdot p) \rceil + 9) \cdot 32]$$

Следовательно, вес транзакции будет следующим:

$$\text{transaction\_weight} = \text{transaction\_size} + \text{transaction\_clawback}$$

Весовое значение блока равно сумме весов транзакций, входящих в его состав, плюс вес майнинговой транзакции.

src/cryptonote_core/blockchain.cpp `create_block_template()`

[20] В истории Bitcoin были моменты, когда объём транзакций становился слишком большим. На этом сайте (https://bitcoinfees.info/) зафиксирован невероятный размер комиссий (до 35 долларов США за одну транзакцию в определённый момент времени).

[21] Следует отметить, что $\log_2(64 \cdot 2) = 7$, а $2 * 7 + 9 = 23$.

### Долгосрочное весовое значение блока

Если разрешить быстро увеличивать размер динамических блоков, размер блокчейна также очень скоро станет неконтролируемым [83]. Чтобы избежать этого, максимальные весовые значения блоков ограничиваются *долгосрочными весовыми значениями блоков*. Помимо обычного весового значения у каждого блока имеется «долгосрочное весовое значение», которое вычисляется на основе фактического среднего долгосрочного весового значения предшествующего блока.[22] Фактическое среднее долгосрочное весовое значение предшествующего блока связано со средними долгосрочными весовыми значениями самых последних 100000 блоков (включая его собственное значение).[23,24]

src/cryptonote_core/blockchain.cpp `update_next_cumulative_weight_limit()`

$$\text{longterm\_block\_weight} = min\{\text{block\_weight}, 1.4 * \text{previous\_effective\_longterm\_median}\}$$
$$\text{effective\_longterm\_median} = max\{300\text{kB}, \text{median\_100000blocks\_longterm\_weights}\}$$

Если размер обычных весовых значений блоков в течение долгого времени остаётся высоким, то понадобится по крайней мере 50000 блоков (примерно 69 дней), чтобы фактическое долгосрочное среднее значение выросло на 40% (именно столько времени необходимо, чтобы долгосрочное весовое значение стало средним).

### Совокупное среднее весовое значение

Объём транзакций может значительно измениться в предельно короткие сроки, особенно в праздники [128]. Для решения этой проблемы Monero предусматривает краткосрочную гибкость весовых значений блоков. Чтобы сгладить переходное изменение, для определения совокупного среднего значения блоков используется среднее значение по обычным весовым значениям последних 100 блоков (включая собственное значение).

`CRYPTONOTE_REWARD_BLOCKS_WINDOW`

$$\text{cumulative\_weights\_median} = max\{300\text{kB}, min\{max\{300\text{kB}, \text{median\_100blocks\_weights}\}, 50 * \text{effective\_longterm\_median}\}\}$$

Следующий блок, добавляемый в блокчейн, ограничивается следующим образом:[25]

[22] Как и в случае со сложностью блоков, обычные и долгосрочные весовые значения блоков вычисляются и сохраняются верификаторами блокчейна, а не включаются в данные блокчейна.

[23] Блоки, созданные до внедрения долгосрочных весовых значений, имеют долгосрочные весовые значения, равные их обычным весовым значениям. Поэтому мы не будем рассматривать подробно то, что связано с генезис-блоком или ранними блоками. Совершенно новая цепочка блоков позволяет делать разумный выбор.

[24] В самом начале составляющая «300 Кбайт» была равна 20 килобайтам, а затем в марте 2016 (версия v2 протокола) выросла до 60 Кбайт [14], а начиная с апреля 2017 (версия v5 протокола) составляет 300 Кбайт [1]. Этот ненулевой «минимальный уровень» средних значений динамического веса блока поспособствовал переходному изменению объёма транзакций, когда абсолютный объём был низок, особенно на ранних этапах внедрения Monero.

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_min_block_weight()`

[25] В версии протокола v8 совокупное медианное значение заменило «M100» (подобную среднюю составляющую). При вычислении штрафов и комиссий, описанных в первой редакции этого отчёта [33], использовалось значение M100.

$$\text{max\_next\_block\_weight} = 2 * \text{cumulative\_weights\_median}$$

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_block_reward()`

Несмотря на то, что максимальное весовое значение блока может вырасти в 100 раз больше фактического среднего долгосрочного весового значения после нескольких сотен блоков, оно не может увеличиться более чем на 40% сверх значения за последние 50000 блоков. Следовательно, рост долгосрочного весового значения блока ограничивается долгосрочными весовыми значениями, а краткосрочные весовые значения могут превысить свои обычные значения.

### 7.3.3 Штраф, налагаемый на вознаграждение за блок

В случае майнинга блоков, размер которых превышает совокупное среднее значение, майнерам приходится платить определённую цену, штраф в форме меньшего вознаграждения за блок. Это означает, что функционально существуют две зоны в пределах максимального весового значения блока: зона без штрафа и зона, облагаемая штрафом. Среднее значение может медленно увеличиваться, что позволяет создавать всё большие блоки без штрафа.

Если весовое значение определённого блока будет больше совокупного среднего значения, то при наличии базового вознаграждения за блок B штраф, налагаемый на вознаграждение за блок, будет следующим:

$$\text{P} = \text{B} * ((\text{block\_weight}/\text{cumulative\_weights\_median}) - 1)^2$$

Таким образом, фактический размер вознаграждения за блок будет[26]

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_block_reward()`

$$\text{B}^{\text{actual}} = \text{B} - \text{P}$$
$$\text{B}^{\text{actual}} = \text{B} * (1 - ((\text{block\_weight}/\text{cumulative\_weights\_median}) - 1)^2)$$

Использование операции ^2 означает, что штрафы субпропорциональны размеру блока. На весовое значение блока на 10% выше предшествующего cumulative_weights_median налагается штраф в размере всего 1% на 50% выше - 25% на 90% выше - штраф в размере 81% и так далее. [19]

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

[26] о реализации конфиденциальных транзакций (RingCt) в версии v4 все суммы передавались простым текстом, а в некоторых ранних версиях разбивались на куски (например → 1000 + 200 + 40 + 4). Чтобы сократить размер майнинговых транзакций, в базовой реализации отбрасывались младшие разряды вознаграждения за блок (всё, что было менее 0,0001 Monero; см. `BASE_REWARD_CLAMP_THRESHOLD`). Это были версии v2-v3. Дополнительный малый разряд не терялся, а использовался в будущих вознаграждениях за блок. В более общем смысле, начиная с версии v2, для вычисления размера вознаграждения за блок просто брался верхний предел реального вознаграждения за блок, который распределялся среди выходов транзакций майнера. Также стоит отметить, что выходы самых ранних транзакций с суммами, прописанными простым текстом, *не разбиваются* на части, не являются «смешиваемыми» выходами, которые потом можно было бы потратить в обычных транзакциях RingCT, создавая кольца из других частей с той же суммой. Правила современного протокола, касающиеся этих «древних» выходов, созданных до RingCT, точно не определены.

src/cryptonote_core/blockchain.cpp `validate_miner_transaction()`

### 7.3.4 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 килобайт данных транзакции.[27] Изначально комиссия просто составляла 0,01 XMR/KiB (добавлена на ранних этапах реализации протокола версии v1) [80], а затем в сентябре 2016 (v3) уменьшилась до 0,002 XMR/KiB[28]

src/cryptonote_core/blockchain.cpp `check_fee()`

В январе 2017 (версия v4 протокола) был добавлен алгоритм вычисления динамической комиссии на KiB [46, 45, 44, 73] was added,[29], а затем вместе со снижением веса транзакций после введения Bulletproofs (v8) расчёт стал производиться не в KiB, а в Кбайтах. Самой важной особенностью алгоритма являлось то, что он не даёт размеру минимально возможной общей комиссии превысить размер вознаграждения за блок (даже при небольшом вознаграждении за блок и больших весовых значениях блоков), что, как считалось, стало причиной нестабильности [98, 47, 59].[30]

#### Алгоритм вычисления комиссии

В основе нашего алгоритма лежит стандартная транзакция [73] весом 3000 байт (подобная транзакции `RCTTypeBulletproof2` с 2 входами и 2 выходами, вес которой, как правило, составляет 2600 байт)[31], а при вычислении комиссии штраф будет смещаться, если среднее значение будет минимальным (самая малая зона, не облагаемая штрафом, 300 Кбайт) [45]. Другими словами, штраф, налагаемый на блок весом 303 Кбайта.

Во-первых, комиссия F, сбалансированная с маргинальным штрафом MP путём добавления весового значения TW к блоку весом BW будет следующей:

$$\mathrm{F} = \mathrm{MP} = \mathrm{B} * (([\mathrm{BW} + \mathrm{TW}]/\mathrm{cumulative\_median} - 1)^2 - \mathrm{B} * ((\mathrm{BW}/\mathrm{cumulative\_median} - 1)^2$$

Определяя коэффициент весового значения блока $\mathrm{WF}_b = (\mathrm{BW}/\mathrm{cumulative\_median} - 1)$, а также коэффициент веса транзакции $\mathrm{WF}_t = (\mathrm{TW}/\mathrm{cumulative\_median})$, упростим:

[27] Этот минимум определяется протоколом консенсуса узлов, а не протоколом блокчейна. Большинство узлов не станут ретранслировать транзакцию другим узлам, если её комиссия будет ниже минимальной (по крайней мере, частично, и только транзакции, «добытые» будут передаваться [45]), но они *будут* принимать новые блоки, содержащие такую транзакцию. В частности, это означает отсутствие необходимости в подержании обратной совместимости в случае с алгоритмами вычисления комиссий. src/cryptonote_core/tx_pool.cpp `add_tx()`

[28] Единица измерения KiB (кибибайт, 1 KiB = 1024 байтам) отличается от Кбайт (килобайт, 1 Кбайт = 1000 байтам).

[29] В апреле 2017 (версия v5 протокола) размер комиссии снизился с 0,002 XMR/KiB до 0,0004 XMR/KiB [1]. В первой редакции данного отчёта описан оригинальный алгоритм расчёта динамической комиссии [33].

[30] Выражаем благодарность архитектору схемы динамического размера блоков и системы комиссий Франсиско Кабаньясу (также известному как ArticMine) за концепции, изложенные в данном Разделе. См. работы [46, 45, 44].

[31] ес базовой транзакции Bitcoin с 1 входом и 2 выходами составляет 250 байт [26] или 430 байт в случае с 2 входами / 2 выходами.

$$\mathrm{F} = \mathrm{B} * (2 * \mathrm{WF}_b * \mathrm{WF}_t + \mathrm{WF}_t^2)$$

При наличии весового значения блока, равного 300 Кбайтам (при совокупном среднем значении, составляющем 300 Кбайт), и нашей стандартной транзакции весом 3000 байт,

$$\begin{aligned}
\mathrm{F}_{\text{ref}} &= \mathrm{B} * (2 * 0 * \mathrm{WF}_t + \mathrm{WF}_t^2) \\
\mathrm{F}_{\text{ref}} &= \mathrm{B} * \mathrm{WF}_t^2 \\
\mathrm{F}_{\text{ref}} &= \mathrm{B} * (\frac{\mathrm{TW}_{\text{ref}}}{\text{cumulative_median}_{\text{ref}}})^2
\end{aligned}$$

Эта комиссия распространяется на 1% зоны, облагаемой штрафом (3000 из 300 000). Эта комиссия также может распространяться на 1% любой зоны, облагаемой штрафом обобщённой стандартной транзакции.

$$\begin{aligned}
\frac{\mathrm{TW}_{\text{ref}}}{\text{cumulative_median}_{\text{ref}}} &= \frac{\mathrm{TW}_{\text{general-ref}}}{\text{cumulative_median}_{\text{general}}} \\
1 &= (\frac{\mathrm{TW}_{\text{general-ref}}}{\text{cumulative_median}_{\text{general}}}) * (\frac{\text{cumulative_median}_{\text{ref}}}{\mathrm{TW}_{\text{ref}}}) \\
\mathrm{F}_{\text{general-ref}} &= \mathrm{F}_{\text{ref}} \\
&= \mathrm{F}_{\text{ref}} * (\frac{\mathrm{TW}_{\text{general-ref}}}{\text{cumulative_median}_{\text{general}}}) * (\frac{\text{cumulative_median}_{\text{ref}}}{\mathrm{TW}_{\text{ref}}}) \\
\mathrm{F}_{\text{general-ref}} &= \mathrm{B} * (\frac{\mathrm{TW}_{\text{general-ref}}}{\text{cumulative_median}_{\text{general}}}) * (\frac{\mathrm{TW}_{\text{ref}}}{\text{cumulative_median}_{\text{ref}}})
\end{aligned}$$

Теперь мы можем масштабировать комиссию на основе реального веса транзакции при известном среднем значении. Так, например, если транзакция попадает на 2% в зону, облагаемую штрафом, комиссия удваивается.

$$\begin{aligned}
\mathrm{F}_{\text{general}} &= \mathrm{F}_{\text{general-ref}} * \frac{\mathrm{TW}_{\text{general}}}{\mathrm{TW}_{\text{general-ref}}} \\
\mathrm{F}_{\text{general}} &= \mathrm{B} * (\frac{\mathrm{TW}_{\text{general}}}{\text{cumulative_median}_{\text{general}}}) * (\frac{\mathrm{TW}_{\text{ref}}}{\text{cumulative_median}_{\text{ref}}})
\end{aligned}$$

Это реорганизуется в комиссию, рассчитываемую в байтах, над чем мы и работаем.

$$\begin{aligned}
f^B_{default} &= \mathrm{F}_{\text{general}}/\mathrm{TW}_{\text{general}} \\
f^B_{default} &= \mathrm{B} * (\frac{1}{\text{cumulative_median}_{\text{general}}}) * (\frac{3000}{300000})
\end{aligned}$$

Если объём транзакции будет ниже среднего значения, не будет никакой причины, чтобы размер комиссии оставался на базовом уровне [73]. По умолчанию мы устанавливаем минимум равный $1/5$.

$$\begin{aligned}
f^B_{min} &= \mathrm{B} * (\frac{1}{\text{cumulative_weights_median}}) * (\frac{3000}{300000}) * (\frac{1}{5}) \\
f^B_{min} &= \mathrm{B} * (\frac{1}{\text{cumulative_weights_median}}) * 0.002
\end{aligned}$$

### Среднее значение комиссии

Выходит так, что совокупное среднее значение комиссий можно использовать для проведения спам-атаки. Повышение краткосрочного среднего значения до максимума (50 x долгосрочное среднее значение) позволяет злоумышленнику использовать минимальные комиссии, поддерживая высокое весовое значение блоков (относительно органичного объёма транзакций) без особых затрат.

Чтобы избежать этого, мы ограничиваем размер комиссий для транзакций, которые войдут в следующий блок, самым низким из доступных средних значений, в результате чего предпочтение более высоким комиссиям будет отдаваться в любом случае.[32]

$$\text{smallest\_median} = max\{300\text{kB}, min\{\text{median\_100blocks\_weights}, \text{effective\_longterm\_median}\}\}$$

src/cryptonote_core blockchain.cpp `check_fee()`

Поддержание более высоких комиссий при росте объёма транзакций также облегчает корректировку краткосрочного среднего значения и гарантирует, что транзакции не останутся ожидать своей очереди, так как майнеры, вероятнее всего, введут их в зону, облагаемую штрафом.

Следовательно, фактическая минимальная комиссия будет следующей:[33,34]

$$f^{B}_{min-actual} = \text{B} * (\frac{1}{\text{smallest\_median}}) * 0.002$$

src/cryptonote_core blockchain.cpp `get_dynamic_base_fee()`

### Комиссии за проведение транзакций

Как сказал Кабаньяс в рамках своей содержательной презентации по этой теме [45]: «Комиссии говорят майнеру о том, насколько большой штраф [авторы транзакции] готовы заплатить, чтобы их транзакция была включена». Майнеры будут заполнять блоки транзакциями в убывающем порядке сумм комиссий [45] (если предположить, что у всех транзакций будет

[32] Злоумышленник может потратить достаточное количество средств комиссионными по краткосрочному значению, чтобы достигнуть 50*долгосрочного среднего значения. При текущем (на момент написания отчёта) размере вознаграждения за блок, составляющем 2 XMR, «оптимизированный» злоумышленник может повышать краткосрочное среднее значение на 17% каждые 50 блоков и достигнуть верхней границы примерно через 1300 блоков (приблизительно 43 часа), потратив 0,39*2 XMR на блок, и при этом на всё им будет потрачено около 1000 XMR (или приблизительно 65k долларов США по текущему курсу), а после он сможет вернуться к минимальной комиссии. Когда среднее значение комиссии станет равным зоне, не облагаемой штрафом, размер минимальной общей комиссии для заполнения зоны, не облагаемой штрафом, составит 0,004 XMR (примерно 0,26 доллара США по текущему курсу). Если среднее значение комиссии будет равным долгосрочному среднему значению, это в соответствии со сценарием спам-атаки составит 1/50 зоны, не облагаемой штрафом. Следовательно, это будет всего 50x краткосрочного среднего значения, 0,2 XMR на блок (13 долларов США). Это составит 2,88 XMR в день против 144 XMR в день (69 дней, пока долгосрочное среднее значение не вырастет на 40%), чтобы обеспечить поддержку веса каждого блока с 50*долгосрочным средним значением. Затраты в размере 1000 XMR будут оправданными в первом случае, но не во втором. Они сокращаются до 300 XMR плюс 43 XMR на поддержку при последующей эмиссии.

[33] Чтобы проверить, является ли определённая комиссия правильной, мы создаём 2% буфер для $f^{B}_{min-actual}$ при наличии целочисленного переполнения (комиссии вычисляются перед тем, как будет окончательно определён вес транзакций). Это означает, что фактическая минимальная комиссия будет следующей: $0.98 * f^{B}_{min-actual}$.

[34] Исследования по улучшению вычисления минимального размера комиссий продолжаются. [131]

src/cryptonote_core blockchain.cpp `check_fee()`

одинаковый вес), поэтому, чтобы зайти в облагаемую штрафом зону, должно быть множество транзакций с большими комиссиями. Это означает, что «потолок» весового значения блока может быть достигнут только в том случае, если общий размер комиссий станет, по крайней мере, в 3-4 раза больше базового вознаграждения за блок (а в этой точке фактическое вознаграждение за блок будет равно нулю).[35]

Чтобы вычислить комиссии за включение транзакции, основной вариант реализации кошелька Monero использует мультипликаторы «приоритета». «Медленная» транзакция напрямую использует минимальную комиссию, а «нормой» является комиссия (5x), заданная по умолчанию, если все транзакции используют «быстрый» (25x) вариант, они могут достичь 2,5% штрафной зоны, а блок, содержащий «супер срочные» (1000x) транзакции, может на 100% попасть в штрафную зону.

src/wallet/wallet2.cpp `get_fee_multiplier()`

Одним из важных последствий введения динамических весовых значений блоков является то, что средний размер общих комиссий за блок будет на порядок величины меньше или, по крайней мере, таким же, как размер вознаграждения за блок (общий размер комиссий может быть равен размеру базового вознаграждения за блок примерно в 37% штрафной зоны [68.5% максимального весового значения блока], в то время как штраф будет составлять 13%). Транзакции, конкурирующие за включение в блок, за которые предлагается более высокая комиссия, способствуют расширению пространства блока и снижению комиссий.[36] Этот механизм взаимодействия представляет собой серьёзную контрмеру против известной угрозы «эгоистичного майнинга» [59].

### 7.3.5 Последующая эмиссия

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

Весовые значения блоков в среднем стабилизируются для всех транзакций, попадающих в сеть, а у авторов транзакций более не будет причины соревноваться, используя комиссии за

[35] Предельный штраф за последние байты, включаемые в блок, сможет рассматриваться как «транзакция», сопоставимая с остальными транзакциями. Чтобы группа транзакций могла выкупить это место для проведения транзакций у майнера, комиссии каждой отдельно взятой транзакции должны быть выше размера штрафа, так как в том случае, если размер хотя бы одной из комиссий будет ниже, майнер сохранит предельное значение вознаграждения. Это последнее предельное вознаграждение, если блок будет заполнен небольшими транзакциями, требует для покупки, по крайней мере, черырёхкратного базового вознаграждения за блок по общей сумме комиссий. Если вес транзакций доходит до максимума (50% минимальной зоны, не облагаемой штрафом, то есть 150 Кбайт), то, если среднее значение сводится к минимуму (300 Кбайт), то для включения последней предельной транзакции потребуется, по крайней мере, трёхкратный размер общей комиссии.

[36] Поскольку размер вознаграждения за блок со временем снижается, а среднее значение увеличивается из-за роста распространения (теоретически), размер комиссий должен стабильно уменьшаться. С точки зрения «реальной покупательной способности» стоимость включения транзакций будет наименее затронута, если цена Monero вырастет вследствие более широкого распространения валюты и экономической дефляции.

включение транзакций сверх минимума, который станет нулевым согласно изложенному в подпункте 7.3.4.

Это создаёт нестабильную и небезопасную ситуацию. У майнеров практически не будет совершенно 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.[37] Если минимальный размер комиссий будет задан как ненулевой, то угроза «эгоистичного майнинга» [59] станет вполне реальной [47].

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

src/cryptonote_basic/cryptonote_basic_impl.cpp `get_block_reward()`

$$
\begin{aligned}
0.6 &> ((L - M) >> 19)/10^{12} \\
\mathrm{M} &> \mathrm{L} - 0.6 * 2^{19} * 10^{12} \\
\mathrm{M}/10^{12} &> \mathrm{L}/10^{12} - 0.6 * 2^{19} \\
\mathrm{M}/10^{12} &> 18,132,171.273709551615
\end{aligned}
$$

то блокчейн Monero никогда не войдёт в состояние так называемой «последующей эмиссии» (emission tail) и вознаграждение в размере 0,6 XMR (0,3 XMR в минуту) будет выплачиваться по-прежнему.[38] Сначала это соответствует примерно 0,9% годовой инфляции и снижается впоследствии.

### 7.3.6 Майнинговая транзакция: `RCTTypeNull`

Майнер блока может заявить своё право на комиссии с транзакций, включаемых в него, и создавать новые деньги в форме вознаграждения за блок. Механизм, позволяющий сделать это, называется «майнинговой транзакцией» (также известной как coinbase-транзакция), подобной обычной транзакции.[39]

src/cryptonote_core/cryptonote_tx_utils.cpp `construct_miner_tx()`

Сумма в выходе (выходах) транзакции майнера должна превышать сумму комиссий транзакций и вознаграждения за блок, и она указывается простым текстом.[40] На месте данных

[37] В случае фиксированной окончательной эмиссии и фиксированного весового значения блоков, как у Bitcoin, ситуация также будет нестабильной. [47]

[38] Начало последующей эмиссии Monero запланировано на май 2022 [17]. Предельное значение денежной массы L будет достигнуто в мае 2024, но, так как эмиссия более не будет зависеть от денежной массы, это не будет иметь значения. В силу доказательства диапазона Monero в одном выходе будет невозможно отправлять сумму, превышающую значение L, даже если кто-то решит накопить больше этого (и с учётом того, что программное обеспечение кошелька такого пользователя позволит делать подобное).

[39] В определённый момент майнинговые транзакции можно было строить в формате урезанных транзакций, которые могли включать в себя некоторые компоненты обычных транзакций (RingCT). Проблемы были устранены в версии v12 после публикации этого хакерского отчёта: [8].

[40] В текущей версии майнеры могут запросить меньше вознаграждение, чем составляет фактическое вычисленное вознаграждение за блок. Остаток отправляется обратно в график эмиссии для будущих майнеров.

выхода записывается высота блока (то есть «Я запрашиваю вознаграждение за блок и комиссии за блок n»).

Факт владения выходом (выходами) майнера указывается путём назначения стандартного адреса[41] с соответствующим публичным ключом транзакции, который сохраняется в поле дополнительных данных. Средства блокируются и их нельзя потратить ещё в течение 60 блоков после публикации [21].[42]

С тех пор как в январе 2017 (версия v4 протокола) [16] 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 выходы обычных транзакций, смешивая их с другими выходами в кольцах MLSAG как обычных, так и майнинговых транзакций.

src/blockchain_db/blockchain_db.cpp `add_transaction()`

Верификаторы блокчейна сохраняют обязательство по сумме каждой майнинговой транзакции блока, созданного после реализации RingCT, и размер каждого обязательства составляет 32 байта.

## 7.4 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 генезис-блок (см. Приложение C)). Следующий блок содержит ссылку на предыдущий блок в форме ID (идентификатора) блока.

src/cryptonote_core/cryptonote_tx_utils.cpp `generate_genesis_block()`

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).[43]

$$\text{Block ID} = \mathcal{H}_n(\text{Block header}, \text{Merkle root}, \#\text{transactions} + 1)$$

src/cryptonote_basic/cryptonote_format_utils.cpp `get_block_hashing_blob()`

Чтобы создать новый блок, необходимо вычислить хеши доказательства работы, изменяя значение нонса, которое хранится в заголовке блока, до тех пор, пока не будет выполнено

[41] Выход майнинговой транзакции теоретически может быть отправлен на подадрес и/или использоваться в мультиподписи и/или зашифрованном ID платежа. Нам не известно, используются или нет эти возможности в каком-либо из вариантов реализации.

[42] Майнинговая транзакция не может быть заблокирована более чем или менее чем на 60 блоков. Если она будет опубликована на 10 блоке, то будет разблокирована именно на высоте 70, и её можно будет потратить на блоке 70 или позднее.

[43] +1 означает включение майнинговой транзакции.

src/cryptonote_core/blockchain.cpp `is_tx_spendtime_unlocked()`

целевое условие сложности.[44] Доказательство работы и ID блока хешируют одну и ту же информацию, но используют разные хеш-функции. Блоки создаются (при $(PoW_{output} * difficulty) > 2^{256} - 1$) путём циклического изменения нонса и пересчёта

$$PoW_{output} = \mathcal{H}_{PoW}(\text{Block header}, \text{Merkle root}, \#\text{transactions} + 1)$$

`get_block_longhash()`

src/cryptonote_basic/cryptonote_format_utils.cpp `calculate_block_hash()`

### 7.4.1 ID транзакции

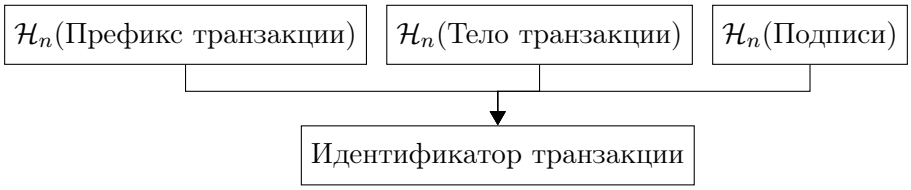
Идентификаторы транзакций похожи на сообщения, подписанные MLSAG-подписью входов (см. подпункт 6.2.2), но также включают в себя MLSAG подписи.

Хешируется следующая информация:

- Префикс транзакции = {версия эры транзакции (то есть RingCT = 2), входы {офсеты ключей, образы ключей}, выходы {одноразовые адреса}, дополнительные данные {публичный ключ транзакции, ID платежа или зашифрованный ID платежа, прочее}}
- Тело транзакции = {тип подписи (`RCTTypeNull` или `RCTTypeBulletproof2`), размер комиссии за транзакцию, обязательства по псевдовыходам, ecdhInfo (зашифрованные или указанные простым текстом суммы), обязательства по выходам}
- Подписи = {MLSAGs, доказательства диапазона}

src/cryptonote_basic/cryptonote_format_utils.cpp `calculate_transaction_hash()`

На следующей диаграмме черными стрелками показаны хеши входов.




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 включает $\mathcal{H}_n(\text{Подписи}) \to 0$, всегда будет уникальным, что упрощает поиск ID.

### 7.4.2 Дерево Меркла

Некоторые пользователи могут пожелать выбросить ненужные данные из своей копии блокчейна. Например, когда вы проверяете доказательства диапазона и подписи входов некоторых

[44] В случае с Monero типичный майнер (по данным https://monerobenchmarks.info/) может выдавать менее 50000 хешей в секунду, то есть менее 6 миллионов хешей на блок. Это означает, что переменная нонса не обязательно должна быть такой величины. Размер нонса Monero составляет 4 байта (максимум 4,3 миллиарда), и было бы странно, если бы какому-то майнеру понадобились все разряды.

транзакций, единственной причиной сохранения этой информации по подписям является возможность самостоятельной верификации транзакции пользователями, получившими её от вас.

Чтобы облегчить «обрезание» данных транзакций, а также для их более общей организации внутри блока, нами используется дерево Меркла [92], которое является простым двоичным деревом хешей. Любая ветвь дерева Меркла может быть обрезана, если вы сохраните её корневой хеш.[45]

src/crypto/tree-hash.c `tree_hash()`

На рисунке 7.1 схематически показан пример дерева Меркла, в основе которого лежат четыре обычные и одна майнинговая транзакция.[46]

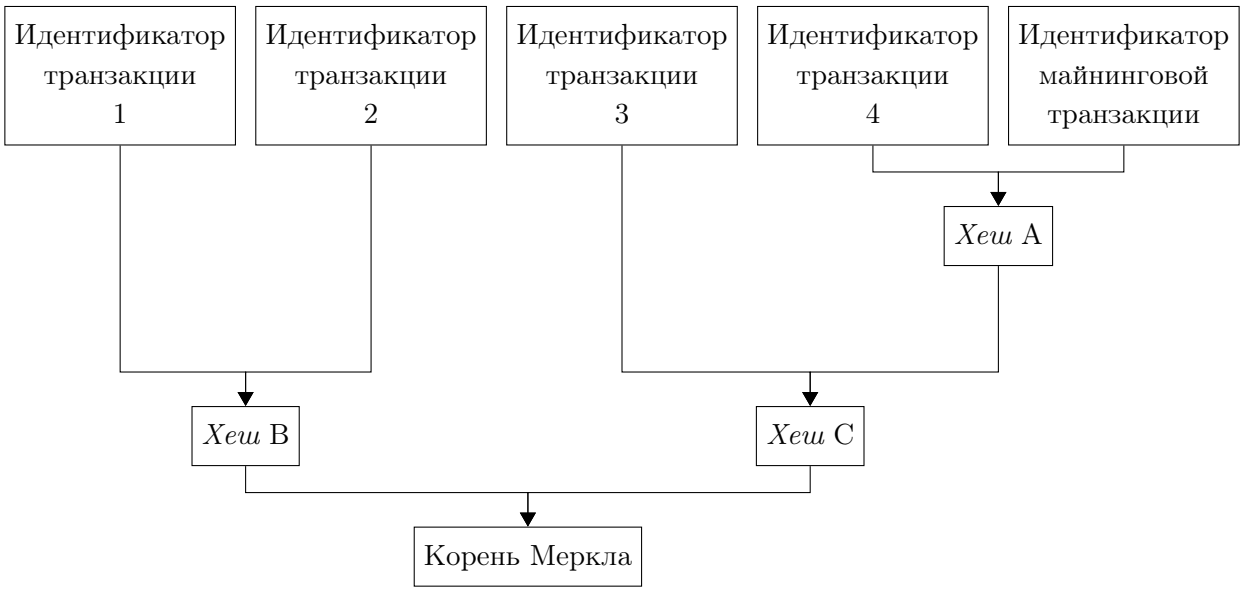



*Рисунок 7.1: Дерево Меркла*

Как видно, корень Меркла ссылается на все включённые транзакции.

[45] Первый известный метод обрезания появился в версии v0.14.1 базовой реализации Monero (март 2019, что совпадает с версией v10 протокола). После верификации транзакции полные узлы могут удалить все данные подписей (включая Bulletproofs, MLSAGs и обязательства по псевдо выходам), но сохранить $\mathcal{H}_n$(Подписей) для вычисления ID транзакции. Но делают это только с 7/8 всех транзакций, поэтому каждая транзакция сохраняется, по крайней мере, 1/8 полных узлов сети. Это сокращает место, необходимо для хранения блокчейна, примерно на 2/3. [55]

[46] Ошибка в коде вычисления дерева Меркла стала причиной серьёзной, но очевидно не критичной атаки на Monero, которая была произведена 4 сентября 2014 [86].

### 7.4.3 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 нами кратко описано содержание блоков. Пример реального блока приводится в Приложении B.

- Заголовок блока:
  - **Старшая версия**: Использовалась для отслеживания хардфорков (изменений протокола).
  - **Младшая версия**: Ранее использовалась для голосования, а теперь просто для отображения старшей версии.
  - **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.

    src/cryptonote_core/blockchain.cpp `check_block_timestamp()`
  - **Идентификатор предыдущего блока**: Ссылка на предыдущий блок. Это важная характеристика блокчейна.
  - **Нонс**: 4-байтовое целое число, которое майнеры перебирают снова и снова, пока хэш доказательства работы (PoW) не достигнет целевого уровня сложности. Любой желающий в качестве проверки может повторно пересчитать полученный хэш доказательства работы.
- Майнинговая транзакция: Распределяет вознаграждение за блок и комиссию за транзакции между майнерами блока.
- Идентификаторы транзакции: Ссылаются на не майнинговые транзакции, добавленные в блокчейн вместе с этим блоком. Идентификаторы транзакций могут в сочетании с идентификаторами майнинговой транзакции использоваться для вычисления корня Меркла, а также для вычисления фактических транзакций, где бы те не хранились.

Помимо данных каждой транзакции (см. подпункт 6.3) мы сохраняем следующую информацию:

- Старшую и младшую версии: переменные целые числа $\leq 9$ байтам.
- Временную метку: переменное целое число $\leq 9$ байтам.
- Идентификатор предыдущего блока: 32 байта.
- Нонс: Размер нонса составляет 4 байта, но фактический размер может быть увеличен за счёт дополнительного нонса дополнительного поля майнинговой транзакции[47]

[47] В каждой транзакции есть «дополнительное» поле, которое может содержать более или менее произвольные данные. Если майнеру нужен более широкий, чем 4 байта, диапазон нонсов, он может добавить или изменить данные в дополнительном поле своей майнинговой транзакции, чтобы «расширить» размер нонса. [81]

- Майнинговую транзакцию: 32 байта на одноразовый адрес, 32 байта на публичный ключ транзакции (+ 1 байт для «дополнительного» тега), а также переменные целые числа для времени разблокировки, высоты и суммы. После загрузки блокчейна нам также понадобится 32 байта для хранения обязательства по суммам $C = 1G + aH$ (касается только сумм майнинговых транзакций, созданных после реализации RingCT).
- Идентификаторы транзакций: по 32 байта на каждый.

# Часть II

# Расширения

# глава 8

# Доказательства знания в транзакциях Monero

Monero — это валюта, и, как и в случае с любой другой валютой, процесс её использования сложен. Начиная со сферы корпоративного бухгалтерского учёта и заканчивая биржевой торговлей и судебными разбирательствами, найдутся самые разные заинтересованные стороны, которым может понадобиться подробная информация о совершённых транзакциях.

Как узнать наверняка, что деньги поступили от конкретного человека? Или доказать, что вы действительно отправили кому-то определённый выход или транзакцию, несмотря на утверждения об обратном? Отправителей и получателей в публичном реестре Monero распознать невозможно. Как доказать, что у вас имеется определённая сумма денег, не раскрывая своих приватных ключей? Суммы в Monero полностью скрыты от внешних наблюдателей.

Нами рассматриваются несколько типов доказательств, используемых в транзакциях, некоторые из которых реализованы в Monero и используются встроенными инструментами кошельков. Также нами предлагается концепция аудита полного баланса на счету физического лица или организации без утечки какой-либо информации относительно будущих возможных транзакций.

## 8.1 Доказательства в транзакциях Monero

Доказательства, используемые в транзакциях Monero, находятся в процессе обновления[103]. Все реализованные на данный момент доказательства имеют версию 1 и не поддерживают разделения доменов. Мы описываем только самые продвинутые доказательства, независимо

того, реализованы ли они на данный момент, планируются для реализации при выходе последующих версий [104] или же являются лишь гипотетическими доказательствами, которые уже могли быть, а могли и не быть реализованы (см. подпункты 8.1.5 [130] и 8.2.1).

### 8.1.1 Многоосновные доказательства в транзакциях Monero

Есть некоторые детали, которые необходимо знать, чтобы двигаться дальше. Большинство доказательств транзакций Monero используют многоосновные доказательства (вспомните подпункт 3.1). Везде, где уместно, разделителем домена будет $T_{txprf2} = \mathcal{H}_n(\text{“TXPROOF\_V2”})$.[1] Подписываемым сообщением, как правило, будет $\mathfrak{m} = \mathcal{H}_n(\texttt{tx\_hash, message})$ (если не будет указано иное), где хеш `tx_hash` является идентификатором соответствующей транзакции (см. подпункт 7.4.1), и `сообщение` будет опциональным. Доказывающие или третьи стороны смогут использовать его, чтобы гарантировать, что у доказывающей стороны на самом деле имеется доказательство и оно не было украдено.

src/wallet/wallet2.cpp `get_tx_proof()`

Доказательства шифруются при помощи кодировки base-58, схемы кодирования двоичного текста, впервые использованной в Bitcoin [28]. Верификация этих доказательств всегда подразумевает сначала их расшифровку из кодировки base-58 обратно в двоичную. Следует отметить, что верификаторам требуется доступ к блокчейну, чтобы они могли использовать ссылки на ID транзакций для получения информации, например, данных одноразовых адресов.[2]

Структура добавления префиксов к ключам в доказательствах является неравномерной из-за того, что в некоторых обновлениях она не была использована. Запросы с 2-основными доказательствами «версии 2» транслируются в этом формате, где, если «базовый ключ 1» является $G$, то его место в запросе заполняется 32 нулевыми байтами,

$$c = \mathcal{H}_n(\mathfrak{m}, \text{public key 2, proof part 1, proof part 2, } T_{txprf2}, \text{public key 1, base key 2, base key 1})$$

### 8.1.2 Доказательство создания входа транзакции (`SpendProofV1`)

Предположим, нами была создана транзакция, и мы хотим доказать её. Очевидно, что если переделать подпись по входу транзакции для нового сообщения, у любого верификатора не останется выбора, кроме как прийти к выводу, что нами был создан оригинал. Переделка *всех* подписей по входам транзакции означает, что мы создали всю транзакцию (вспомните подпункт 6.2.2), или же самое малое полностью «профинансировали» её.[3]

[1] Подобно тому как было описано в подпункте 3.6, хеш-функции должны быть разделены по доменам путём добавления к ним префикса в виде тега. Используемый в настоящее время вариант реализации доказательств транзакций не предполагает разделения по доменам, поэтому все теги, которые указываются в данной главе, *пока не реализованы.*

[2] Идентификаторы транзакций обычно передаются отдельно от доказательств.

[3] Как станет понятно из Главы 11, совсем не обязательно тот, кто создаёт одну подпись по входу, создаёт все подписи по входам.

Так называемое SpendProof содержит переделанные подписи по всем входам транзакции. Что важно, в кольцевых подписях SpendProof повторно используются участники оригинального кольца, что позволяет не допустить идентификации истинного подписанта путём анализа «пересечения» колец.

src/wallet/wallet2.cpp `get_spend_proof()`

Доказательства SpendProof реализованы в Monero, и, чтобы зашифровать такое доказательство для передачи верификаторам, доказывающей стороне необходимо присоединить строку префикса "`SpendProofV1`" к списку подписей. Следует отметить, что кодировка base-58 не применяется к строке префикса и её не требуется шифровать/кодировать, поскольку она должна быть читабельной для человека.

### Доказательство SpendProof

Доказательства SpendProof не используют подписей MLSAG. Они используют оригинальную схему кольцевой подписи Monero, которая использовалась в рамках самого первого протокола транзакций (до введения RingCT) [136].

src/crypto/crypto.cpp `generate_ring_signature()`

1. Вычисляем образ ключа $\tilde{K} = k_\pi^o \mathcal{H}_p(K_\pi^o)$.
2. Генерируем случайное число $\alpha \in_R \mathbb{Z}_l$ и случайные числа $c_i, r_i \in_R \mathbb{Z}_l$ для $i \in \{1, 2, ..., n\}$, за исключением $i = \pi$.
3. Вычисляем
$$c_{tot} = \mathcal{H}_n(\mathfrak{m}, [r_1 G + c_1 K_1^o], [r_1 \mathcal{H}_p(K_1^o) + c_1 \tilde{K}], ..., [\alpha G], [\alpha \mathcal{H}_p(K_\pi^o)], ..., \text{etc.})$$
4. Определяем реальный запрос
$$c_\pi = c_{tot} - \sum_{i=1, i \neq \pi}^{n} c_i$$
5. Определяем $r_\pi = \alpha - c_\pi * k_\pi^o \pmod{l}$.

Подпись будет следующей: $\sigma = (c_1, r_1, c_2, r_2, ..., c_n, r_n)$.

### Верификация

Чтобы верифицировать доказательство SpemdProof конкретной транзакции, верификатору, воспользовавшись информацией из соответствующей эталонной транзакции (например, образами ключей и значениями смещения выходов для получения одноразовых адресов от других транзакций), необходимо подтвердить, что все кольцевые подписи являются действительными.

src/wallet/wallet2.cpp `check_spend_proof()`

1. Вычисляем
$$c_{tot} = \mathcal{H}_n(\mathfrak{m}, [r_1 G + c_1 K_1^o], [r_1 \mathcal{H}_p(K_1^o) + c_1 \tilde{K}], ..., [r_n G + c_n K_n^o], [r_n \mathcal{H}_p(K_n^o) + c_n \tilde{K}])$$
2. Проверяем равенство
$$c_{tot} \stackrel{?}{=} \sum_{i=1}^{n} c_i$$

#### Почему это работает

Следует отметить, что это та же схема, что и в случае с bLSAG (см. подпункт 3.4), при которой присутствует только один участник кольца. Чтобы добавить ложного участника, а не решать превращать запрос $c_{\pi+1}$ в новый хеш запроса, участник добавляется в оригинальный хеш. Поскольку уравнение

$$c_s = c_{tot} - \sum_{i=1, i \neq s}^{n} c_i$$

заведомо остаётся действительными для любого индекса $s$, верификатор никак не сможет идентифицировать реальный запрос. Более того, не зная $k_\pi^o$, доказывающая сторона никогда не смогла бы правильно определить $r_\pi$ (кроме как с ничтожной вероятностью).

### 8.1.3 Создание доказательства по выходу транзакции (`OutProofV2`)

Теперь предположим, что мы отправляем какие-то деньги (выход) и хотим доказать это. Выходы транзакций включают в себя три компонента: адрес получателя, отправляемую сумму и приватный ключ транзакции. Суммы зашифрованы, поэтому, чтобы начать, нам, по сути, требуются только адрес и приватный ключ транзакции. Любой, кто удалит или потеряет свой приватный ключ транзакции не сможет создать доказательства OutProof, поэтому в данном контексте доказательства OutProof представляются наименее надёжными из всех доказательств транзакций Monero.[4]

В данном случае наша задача состоит в том, чтобы показать, что одноразовый адрес был создан на основе адреса получателя, и позволить верификаторам восстановить обязательство по выходу. Мы можем сделать это, доказав общий секрет отправителя и получателя $rK^v$, а затем доказав, что мы создали его и что он соответствует публичному ключу транзакции и адресу получателя, подписав 2-основную подпись (см. подпункт 3.1) по базовым ключам $G$ и $K^v$. Верификаторы могут использовать общий секрет для проверки получателя (см. подпункт 4.2), расшифровки суммы (см. подпункт 5.3) и восстановления обязательства по выходу (см. подпункт 5.3). Мы даём подробную информацию как по обычным адресам, так и по подадресам.

src/wallet/wallet2.cpp `check_tx_proof()`

#### Доказательство OutProof

Чтобы сгенерировать доказательство по выходу, направленному по адресу $(K^v, K^s)$ или подадресам $(K^{v,i}, K^{s,i})$, при наличии приватного ключа транзакции $r$, где общим секретом отправителя и получателя является $rK^v$, вспомним, что публичным ключрм транзакции, сохранённым в данных транзакции, будет либо $rG$, либо $rK^{s,i}$,i, в зависимости от того, использует или нет получатель подадрес (см. подпункт 4.3).

src/crypto/crypto.cpp `generate_tx_proof()`

[4] Доказательство OutProof можно рассматривать в качестве доказательства того, что выход поступает от доказывающей стороны. Соответствующие доказательства InProof (см. подпункт 8.1.4) показывают выходы, поступающие на адрес доказывающей стороны.

1. Генерируем случайное число $\alpha \in_R \mathbb{Z}_l$ и вычисляем:
   (a) *для обычного адреса*: $\alpha G$ and $\alpha K^v$
   (b) *для подадреса*: $\alpha K^{s,i}$ and $\alpha K^{v,i}$
2. Вычисляем запрос
   (a) *для обычного адреса*:[5]
   $$c = \mathcal{H}_n(\mathfrak{m}, [rK^v], [\alpha G], [\alpha K^v], [T_{txprf2}], [rG], [K^v], [0])$$
   (b) *для подадреса*:
   $$c = \mathcal{H}_n(\mathfrak{m}, [rK^{v,i}], [\alpha K^{s,i}], [\alpha K^{v,i}], [T_{txprf2}], [rK^{s,i}], [K^{v,i}], [K^{s,i}])$$
3. Определяем ответ[6] $r^{resp} = \alpha - c * r$.
4. Подпись будет следующей $\sigma^{outproof} = (c, r^{resp})$.

Доказывающая сторона может сгенерировать целую группу доказательств OuProof и отправить все их верификатору. Он присоединит строку префикса "`OutProofV2`" к списку доказательств, где каждый пункт (зашифрованный с использованием кодировки base-58) будет включать в себя общий секрет отправителя и получателя $rK^v$ (или $rK^{v,i}$ в случае использования подадреса) и соответствующую подпись $\sigma^{outproof}$. Предполагается, что верификатору известен соответствующий адрес для каждого доказательства.

src/wallet/wallet2.cpp `get_tx_proof()`

### Верификация

1. Вычисляем запрос
   (a) *для обычного адреса*:
   $$c' = \mathcal{H}_n(\mathfrak{m}, [rK^v], [r^{resp}G + c * rG], [r^{resp}K^v + c * rK^v], [T_{txprf2}], [rG], [K^v], [0])$$
   (b) *для подадреса*:
   $$c' = \mathcal{H}_n(\mathfrak{m}, [rK^{v,i}], [r^{resp}K^{s,i} + c*rK^{s,i}], [r^{resp}K^{v,i} + c*rK^{v,i}], [T_{txprf2}], [rK^{s,i}], [K^{v,i}], [K^{s,i}])$$

   src/crypto/crypto.cpp `check_tx_proof()`
2. Если $c = c'$, значит, доказывающей стороне известен $r$, а $rK^v$ является действительным общим секретом между $rG$ и $K^v$ (за исключением ничтожной вероятности).
3. Верификатор должен проверить адрес получателя, который может использоваться для создания одноразового адреса соответствующей транзакции (вычисления будут одинаковыми как для обычных адресов, так и для подадресов)
   $$K^s \stackrel{?}{=} K_t^o - \mathcal{H}_n(rK^v, t)$$

   src/wallet/wallet2.cpp `check_tx_key_helper()`

[5] В данном случае значение 0 является 32-байтовой кодировкой нулевых байтов.

[6] Из-за ограниченного количества доступных символов, к сожалению, нам пришлось использовать $r$ как для обозначения ответов, так и приватного ключа транзакции. По необходимости, в целях обозначения отличия, будет использоваться верхний индекс 'resp', обозначающий 'response' (ответ).

4. Также необходимо расшифровать сумму в выходе $b_t$, вычислить маску выхода $y_t$ и попытаться восстановить соответствующее обязательство по выходу[7]

$$C_t^b \stackrel{?}{=} y_t G + b_t H$$

### 8.1.4 Доказательство владения выходом (`InProofV2`)

Доказательство OutProof демонстрирует, что доказывающая сторона отправила выход на адрес, в то время как доказательство InProof показывает, что выход был получен на какой-то определённый адрес. Как правило, это другая «сторона» в общем секрете отправителя и получателя $rK^v$. В этот раз доказывающая сторона доказывает знание $k^v$ в $K^v$, а это в сочетании с публичным ключом транзакции $rG$ даёт общий секрет $k^v * rG$.

Как только верификатор получит $rK^v$, он сможет, используя $K^o - \mathcal{H}_n(k^v * rG, t) * G \stackrel{?}{=} K^s$, проверить, принадлежит ли одноразовый адрес адресу доказывающей стороны (см. подпункт 4.2.1). Создавая InProof для всех публичных ключей в блокчейне, доказывающая сторона раскроет все выходы, которыми обладает.

src/wallet/wallet2.cpp `check_tx_proof()`

Прямая передача ключа просмотра верификатору имела бы те же последствия, но как только он получит такой ключ, он сможет идентифицировать принадлежность выходов, которые будут создаваться в будущем. При наличии доказательств InProof верификатор сохраняет контроль над своими приватными ключами, но за счёт времени, необходимого для доказательства (и последующей верификации) принадлежности или отсутствия принадлежности выхода.

#### Доказательство InProof

InProof строится так же, как и OutProof, за тем лишь исключением, что базовыми ключами теперь будут $\mathcal{J} = \{G, rG\}$, публичными ключами $\mathcal{K} = \{K^v, rK^v\}$, а ключом подписания станет $k^v$ вместо $r$.Для ясности понимания мы покажем только этап верификации. Следует отметить, что порядок добавления префиксов к ключам изменяется ($rG$ и $K^v$ меняются местами) в целях обеспечения соответствия роли, исполняемой каждым из ключей.

src/crypto/crypto.cpp `generate_tx_proof()`

Верификатору может быть отправлено сразу множество доказательств InProof, связанных с множеством выходов, имеющимся по одному и тому же адресу. Вначале добавляется

src/wallet/wallet2.cpp `get_tx_proof()`

[7] Действительная подпись OutProof совсем не обязательно будет означать, что получатель является реальным получателем. Доказывающая сторона, действующая со злым умыслом, может сгенерировать случайный ключ просмотра $K'^v$, вычислить $K'^s = K^o - \mathcal{H}_n(rK'^v, t) * G$ и выдать $(K'^v, K'^s)$ в качестве условного получателя. За счёт повторного вычисления обязательства по выходу верификаторы могут быть более уверены в том, что рассматриваемый адрес получателя является легитимным. Тем не менее доказывающая сторона и получатель могут объединить усилия и зашифровать обязательство по выходу, воспользовавшись $K'^v$, в то время как одноразовый адрес будет использовать $(K^v, K^s)$. Поскольку получателю понадобится знание приватного ключа $k'^v$ (предполагается, что сумму в выходе по-прежнему можно будет потратить), это сомнительное свойство при таком уровне жульничества. А почему бы получателю просто не использовать $(K'^v, K'^s)$ (или некоторый другой одноразовый адрес) для всего выхода? Поскольку вычисление $C_t^b$ связано с получателем, мы считаем описанный процесс верификации OutProof адекватным. Другими словами, доказывающая сторона не сможет использовать его для введения верификаторов в заблуждение, не координируя свои действия с получателем.

строка префикса "`InProofV2`", и каждое доказательство (зашифрованное с использованием кодировки base-58) будет включать в себя общий секрет отправителя и получателя $rK^v$ (или $rK^{v,i}$) и соответствующую подпись $\sigma^{inproof}$.

#### Верификация

1. Вычисляем запрос: src/crypto/crypto.cpp `check_tx_proof()`
   (a) *для обычного адреса*:
   $$c' = \mathcal{H}_n(\mathfrak{m}, [rK^v], [r^{resp}G + c * K^v], [r^{resp} * rG + c * k^v * rG], [T_{txprf2}], [K^v], [rG], [0])$$
   (b) *для подадреса*:
   $$c' = \mathcal{H}_n(\mathfrak{m}, [rK^{v,i}], [r^{resp}K^{s,i} + c * K^{v,i}], [r^{resp} * rK^{s,i} + c * k^v * rK^{s,i}], [T_{txprf2}], [K^{v,i}], [rK^{s,i}], [K^{s,i}])$$
2. Если $c = c'$, значит, доказывающей стороне известен $k^v$, а $k^v * rG$ является действительным общим секретом между $K^v$ и $rG$ (за исключением ничтожной вероятности).

#### Доказательство «полноты» владения ключом к одноразовому адресу

Несмотря на то, что доказательство InProof демонстрирует, что одноразовый адрес был построен по определённому адресу (за исключением ничтожной вероятности), это не обязательно означает, что доказывающая сторона может *потратить* выход. Только те, кто может потратить выход, действительно владеют таким выходом.

Доказательство владения при завершении InProof настолько же просто, как и подписание сообщения при помощи ключа траты.[8]

### 8.1.5 Доказательство того, что имеющийся выход не был потрачен в какой-либо транзакции (UnspentProof)

Доказательство того, что выход был потрачен или не потрачен, так же просто, как и создание его образа ключа с многоосновным доказательством по $\mathcal{J} = \{G, \mathcal{H}_p(K^o)\}$ и $\mathcal{K} = \{K^o, \tilde{K}\}$. Несмотря на то, что это, очевидно, работает, верификаторам необходимо знать образ ключа, который так же раскрывает, когда непотраченный выход будет потрачен *в будущем*.

Но мы можем доказать, что выход не был потрачен в определённой транзакции, не раскрывая образа ключа. Более того, мы *полностью* можем доказать, что до настоящего момента выход не был потрачен, расширив это доказательство [130] до всех транзакций, в которые он был включён в качестве участника кольца.[9]

[8] Возможность предоставления такой подписи напрямую в Monero, кажется, невозможно, даже несмотря на то, что доказательства ReserveProof включают её (см. подпункт 8.1.6).

[9] Доказательства UnspentProof не были реализованы в Monero.

В частности, наше доказательство UnspentProof указывает на то, что определённый образ ключа транзакции, находящейся в блокчейне, соответствует или не соответствует определённому одноразовому адресу соответствующего кольца. В данном случае, как мы увидим, доказательства UnspentProof неразрывно связаны с доказательствами InProof.

### Создание доказательства UnspentProof

Верификатор UnspentProof должен знать $rK^v$, общий секрет отправителя и получателя для определённого имеющегося выхода с одноразовым адресом $K^o$ и публичным ключом транзакции $rG$. Ему должен быть известен либо ключ просмотра $k^v$, позволяющий вычислить $k^v * rG$ и проверить $K^o - \mathcal{H}_n(k^v * rG, t) * G \stackrel{?}{=} K^s$, чтобы он знал, что проверяемый выход принадлежит доказывающей стороне (см. подпункт 4.2), либо предоставленный доказывающей стороной общий секрет $rK^v$. Вот здесь и вступают в игру доказательства InProof, поскольку при наличии InProof верификатор может быть уверен в том, что $rK^v$ действительно принадлежит ключу просмотра доказывающей стороны и соответствует имеющемуся у неё выходу. При этом верификатору не требуется приватного ключа просмотра.

Перед проверкой UnspentProof верификатору необходимо узнать образ ключа, который будет проверяться $\tilde{K}_?$, и он также проверяет, содержит ли соответствующее кольцо одноразовый адрес $K^o$ выхода, имеющегося у доказывающей стороны. Затем верификатор вычисляет частичный образ «траты» $\tilde{K}^s_?$.

$$\tilde{K}^s_? = \tilde{K}_? - \mathcal{H}_n(rK^v, t) * \mathcal{H}_p(K^o)$$

Если проверяемый образ ключа был создан на основе $K^o$, то полученной точкой будет $\tilde{K}^s_? = k^s * \mathcal{H}_p(K^o)$.

### Доказательство UnspentProof

Доказывающая сторона создаёт два многоосновных доказательства (см. подпункт 3.1). Её адресом, которому принадлежит рассматриваемый выход, будет $(K^v, K^s)$ или $(K^{v,i}, K^{s,i})$.[10]

1. 3-основное доказательство, где подписывающим ключом является $k^s$, а
$$\mathcal{J}_3^{unspent} = \{[G], [K^s], [\tilde{K}^s_?]\}$$
$$\mathcal{K}_3^{unspent} = \{[K^s], [k^s * K^s], [k^s * \tilde{K}^s_?]\}$$
2. 2-основное доказательство, где подписывающим ключом является $k^s * k^s$, а
$$\mathcal{J}_2^{unspent} = \{[G], [\mathcal{H}_p(K^o)]\}$$
$$\mathcal{K}_2^{unspent} = \{[k^s * K^s], [k^s * k^s * \mathcal{H}_p(K^o)]\}$$

[10] оказательства UnspentProof одинаково создаются как для обычных адресов, так и для подадресов. Для этого требуется полный ключ траты конкретного подадреса, например, $k^{s,i} = k^s + \mathcal{H}_n(k^v, i)$ (см. подпункт 4.3).

Наряду с доказательствами $\sigma_3^{unspent}$ и $\sigma_2^{unspent}$ доказывающая сторона должна передать публичные ключи $k^s * K^s$, $k^s * \tilde{K}_?^s$ и $k^s * k^s * \mathcal{H}_p(K^o)$.

**Верификация**

1. Подтверждаем легитимность $\sigma_3^{unspent}$ и $\sigma_2^{unspent}$.
2. Убеждаемся в том, что публичный ключ $k^s * K^s$ использовался в обоих доказательствах.
3. Проверяем, одинаковы ли $k^s * \tilde{K}_?^s$ и $k^s * k^s * \mathcal{H}_p(K^o)$. Если да, то выход был потрачен, а если нет, то он не был потрачен (за исключением ничтожной вероятности).

**Почему это работает**

Этот кажущийся обходным подход не даёт верификатору узнать $k^s * H_p(K^o)$ непотраченного выхода, который он мог бы использовать в сочетании с $rK^v$ для вычисления реального образа ключа. При этом у него будет уверенность в том, что проверенный образ ключа не соответствует выходу.

Доказательство $\sigma_2^{unspent}$ может повторно использоваться для любого количества доказательств UnspentProof, касающихся одного и того же выхода, и даже если он будет фактически потрачен, понадобится только одно (то есть, доказательства UnspentProofs также могут использоваться для того, чтобы продемонстрировать, что выход был потрачен). Использование доказательств UnspentProofs во всех кольцевых подписях, в которых был отмечен определённый непотраченный выход, не должно быть затратным с точки зрения требуемых вычислений. Выход, вероятно со временем будет включён в качестве ложного участника в 11 (текущий размер кольца) различных колец.

### 8.1.6 Доказательство наличия минимального количества непотраченных средств по адресу (`ReserveProofV2`)

Несмотря на снижение уровня анонимности при раскрытии образа ключа выхода, когда он ещё не потрачен, это всё же полезный метод, и он был реализован в Monero [126] до того, как были изобретены доказательства UnspentProof [130]. Так называемое доказательство 'ReserveProof' используется в Monero для доказательства того факта, что по адресу имеется минимальная сумма денег, путём создания образов ключей для некоторых непотраченных выходов.

src/wallet/wallet2.cpp `get_reserve_proof()`

Если говорить более конкретно, при наличии минимального баланса доказывающая сторона может найти достаточное количество выходов, чтобы покрыть его, и продемонстрирует такое наличии при помощи InProofs, создавая образы ключей для таких выходов, и доказывает, что в основе действительно лежат выходы, при помощи 2-основных доказательств (используя

различный формат добавления префиксов к ключам). А затем она доказывает, что ей известны приватные ключи траты, используемые в обычных подписях Шнорра (ключей может быть несколько, если выходы находятся по разным подадресам). Верификатор может проверить, появлялись или нет образы ключей в блокчейне, и если нет, то соответствующие выходы должны быть непотраченными.

### Доказательство ReserveProof

Все «поддоказательства» в ReserveProof используются для подписания иных сообщений, не тех, которые подписываются другими доказательствами (например, OutProof, InProof или SpendProofs). В этот раз это $\mathfrak{m} = \mathcal{H}_n(\texttt{message}, \texttt{address}, \tilde{K}_1^o, ..., \tilde{K}_n^o)$, где `address` является зашифрованной формой (см. работу [5]) обычного адреса доказывающей стороны $(K^v, K^s)$, а образы ключей соответствуют непотраченным выходам, которые будут включены в доказательство.

1. У каждого выхода есть доказательство InProof, которое демонстрирует, что данный выход находится по адресу доказывающей стороны (или по одному из её подадресов).

2. Образ ключа каждого выхода подписывается 2-основным доказательством, где запрос форматируется подобным образом: src/crypto/crypto.cpp `generate_ring_signature()`
$$c = \mathcal{H}_n(\mathfrak{m}, [rG + c * K^o], [r\mathcal{H}_p(K^o) + c * \tilde{K}])$$

3. Каждый адрес (и подадрес), по которому находится по крайней мере один выход, имеет обычную подпись Шнорра (см. подпункт 2.3.5), и запрос (который будет одинаковым как для обычных адресов, так и для подадресов) выглядит так: src/crypto/crypto.cpp `generate_signature()`
$$c = \mathcal{H}_n(\mathfrak{m}, K^{s,i}, [rG + c * K^{s,i}])$$

Чтобы отправить доказательство ReserveProof кому-то ещё, доказывающей стороне необходимо добавить строку префикса “`ReserveProofV2`" в два списка, зашифрованных с использованием кодировки base-58 (например, “`ReserveProofV2`, list 1, list 2"). Каждое доказательство в списке 1 связано с определённым выходом и содержит хеш соответствующей транзакции (см. подпункт 7.4.1), индекс выхода для данной транзакции (см. подпункт 4.2.1), соответствующий общий секрет $rK^v$, его образ ключа, его $\sigma^{inproof}$ InProof и его доказательство образа ключа. В списке 2 содержатся адреса, по которым находятся эти выходы, наряду с их подписями Шнорра. src/wallet/wallet2.cpp `get_reserve_proof()`

### Верификация

1. Проверяем отсутствие образов ключей ReserveProof в блокчейне. src/wallet/wallet2.cpp `check_reserve_proof()`

2. Проверяем InProof для каждого выхода и что один из предоставленных адресов принадлежит каждому из них.

3. Проверяем 2-основные подписи образов ключей.

4. Используем общие секреты отправителя и получателя для расшифровки сумм в выходах (см. подпункт 5.3).

5. Проверяем подпись каждого адреса.

Если всё будет верно, то у доказывающей стороны должна иметься непотраченная, по крайней мере, общая сумма, указанная в выходах ReserveProof (за исключением ничтожной вероятности).[11]

## 8.2 Концепция аудита Monero

В США большинство компаний проходит ежегодный аудит своей финансовой отчётности [129], которая включает в себя отчёт о доходах, балансовый отчёт и баланс оборотных средств. Первые два документа, по сути, представляют собой большую часть внутренних записей компании, в то время как в последнем указывается каждая транзакция, влияющая на общее количество денег, имеющихся на данный момент у компании. Криптовалюты являются цифровыми деньгами, поэтому любой аудит баланса оборотных криптовалютных средств пользователя также будет связан с транзакциями, данные которых сохраняются в блокчейне.

Первая задача лица, проходящего аудит, состоит в указании всех выходов, которые у него имеются на данный момент (как потраченных, так и нет). Это можно сделать при помощи доказательств InProof, использующих все адреса. В случае с большим бизнесом подадресов может быть множество, особенно когда речь идёт о ритейлерах, работающих в формате торговых онлайн площадок (см. Главу 10). Создание доказательств InProof для всех транзакций и каждого отдельно взятого подадреса может привести к появлению чудовищных затрат, связанных с вычислениями и хранением данных как со стороны доказывающих сторон, так и верификаторов.

Вместо этого мы можем создавать доказательства InProof только для нормального адреса доказывающей стороны (для всех транзакций). Аудитор использует общие секреты отправителя и получателя для проверки того, принадлежит ли какой-либо из выходов основному адресу доказывающей стороны или связанным с ним подадресам. Как было сказано в подпункте 4.3, для идентификации всех выходов, имеющихся по подадресам основного адреса, достаточно ключа просмотра пользователя.

Чтобы убедиться в том, что доказывающая сторона не вводит в заблуждение аудитора, скрывая обычный адрес для некоторых из её подадресов, она также должна доказать, что все подадреса соответствуют одному из её известных обычных адресов.

[11] Несмотря на то, что доказательства ReserveProof демонстрируют полную принадлежность средств, они не включают в себя доказательств того, что данные подадреса фактически соответствуют обычному адресу доказывающей стороны.

### 8.2.1 Доказательство соответствия адреса и подадресов (SubaddressProof)

Доказательства SubaddressProof демонстрируют, что ключ просмотра адреса может быть использован для идентификации выходов, имеющихся по определённому подадресу.[12]

#### Доказательство SubaddressProof

Доказательства SubaddressProof создаются во многом так же, как доказательства OutProofs и InProofs. В данном случае базовыми ключами являются $\mathcal{J} = \{G, K^{s,i}\}$, публичными ключами - $\mathcal{K} = \{K^v, K^{v,i}\}$, а подписывающим ключом - $k^v$. И вновь для ясности понимания мы покажем только этап верификации.

#### Верификация

Верификатору известен адрес доказывающей стороны $(K^v, K^s)$, подадрес $(K^{v,i}, K^{s,i})$, и $\sigma^{subproof} = (c, r)$ SubaddressProof.

1. Вычисляем запрос
$$c' = \mathcal{H}_n(\mathfrak{m}, [K^{v,i}], [rG + c * K^v], [rK^{s,i} + c * K^{v,i}], [T_{txprf2}], [K^v], [K^{s,i}], [0])$$
2. Если $c = c'$, значит, доказывающей стороне известен $k^v$ для $K^v$, а $K^{s,i}$ в сочетании с этим ключом просмотра даёт $K^{v,i}$ (за исключением ничтожной вероятности).

### 8.2.2 Концепция аудита

Теперь мы готовы получить максимум информации о транзакциях пользователя.[13]

1. Доказывающая сторона собирает список всех своих счетов, и каждый счёт включает в себя обычный адрес и различные подадреса. Он создаёт доказательства SubaddressProof для всех подадресов. Во многом как и в случае с доказательствами ReserveProof, он также создаёт подпись, используя ключ траты каждого адреса и подадреса, демонстрируя тем самым, что у него есть права на трату всех выходов, находящихся по этим адресам.

[12] Доказательства SubaddressProof не были реализованы в Monero.

[13] Данная концепция аудита доступна в случае Monero не полностью. Доказательства SubaddressProof и UnspentProof не реализованы. Доказательства InProof не готовы к оптимизации, связанной с подадресами, что мы уже объясняли. Кроме того, структура, необходимая для лёгкого получения или организации информации, необходимой как доказывающим сторонам, так и верификаторам, отсутствует.

2. Для каждого своего обычного адреса доказывающая сторона генерирует доказательства InProof по всем транзакциям (например, всем публичным ключам транзакций), имеющимся в блокчейне. Это раскрывает аудитору все выходы, которые имеются по адресам доказывающей стороны, так как он может проверить все одноразовые адреса, воспользовавшись общим секретом отправителя и получателя. Благодаря доказательствам SubaddressProof он может убедиться в том, что выходы, имеющиеся по подадресам, будут идентифицированы.[14]

3. Для каждого имеющегося у неё выхода доказывающая сторона генерирует доказательства UnspentProof по всем входам транзакций, где они являются участниками колец. После этого аудитору будет известен баланс доказывающей стороны, и он сможет дальше проверять потраченные выходы.[15]

4. *Как вариант*: для каждой транзакции, где был потрачен выход, доказывающая сторона генерирует доказательство OutProof, чтобы показать аудитору получателя и сумму. Этот шаг возможен только в том случае, если доказывающей стороной был сохранён приватный ключ (или ключи) транзакции.

Важен тот факт, что у доказывающей стороны нет возможности напрямую продемонстрировать происхождение средств. Единственным способом является запрос ряда доказательств от людей, отправивших ей деньги.

1. В отношении транзакции, в которой доказывающей стороне были переданы средства, автор такой транзакции создаёт доказательство SpendProof, демонстрирующее, что именно он отправил эти средства.

2. Также тот, кто передал средства доказывающей стороне, используя идентифицирующий публичный ключ, создаёт подпись. Например, это может быть ключ траты обычного адреса того, кто отправил средства. При помощи доказательства SpendProof вместе с этой подписью вновь подписывается сообщение, содержащее этот идентифицирующий публичный ключ, и это гарантирует, что доказательство SpendProof не было украдено или, по сути, создано кем-то другим.

[14] Этот шаг также может быть выполнен путём предоставления приватных ключей просмотра, но это скажется на общем уровне анонимности.

[15] Как вариант, можно создать доказательства ReserveProof для всех имеющихся выходов. И снова, раскрытие образов ключей непотраченных выходов имеет очевидные последствия с точки зрения анонимности.

# глава 9

# Мультиподписи в Monero

Криптовалютные транзакции необратимы. Если кому-нибудь удастся украсть ваши приватные ключи или каким-то другим образом обмануть вас, деньги могут быть потеряны навсегда. Распределение полномочий, связанных с подписанием транзакций, между людьми может снизить потенциальный риск такого обмана.

Допустим, вы кладёте деньги на общий счёт с компанией, занимающейся вопросами безопасности и отслеживающей подозрительную деятельность, связанную с вашим счётом. Транзакции могут быть подписаны только в том случае, если вы сделаете это вместе с компанией. Если кто-нибудь украдёт ваши ключи, вы сможете сообщить компании о проблеме, и компания остановит процесс подписания транзакций для вашего счёта. Обычно это называют «эскроу» сервисом.[1]

В случае с криптовалютами технология «мультиподписей» используется с целью совместного подписания транзакций при помощи так называемой схемы multisig «M из N». В рамках данной схемы N пользователей объединяется для создания общего ключа, и только M пользователей ($M \leq N$) требуется подписаться с использованием этого ключа. Мы начнём эту главу с объяснения базовых принципов схемы multisig «N из N», перейдём к схеме multisig «N из N» Monero, обобщим её до схемы multisig «M из N», а затем разъясним, как вложить одни multisig-ключи в другие.

[1] Мультиподписи имеют множество применений, начиная с корпоративных счетов и заканчивая подписками на рассылку информации торговыми онлайн площадками.

В данной главе мы *фокусируем* внимание на том, как нам видится работа механизма multisig, исходя из рекомендаций, изложенных в работе [111], а также данных различных исследований, касающихся вариантов эффективной реализации. В сносках мы попытались указать, чем текущая версия реализации отличается от того, что мы описываем.[2] Наши контрибьюторы подробно изучили схему multisig «M из N» и новый подход к вложению multisig-ключей.

## 9.1 Обмен данными с другими подписантами

Создание общих ключей и общих транзакций требует передачи секретной информации между людьми, которые могут находиться в любой точке земного шара. Чтобы защитить эту информацию от наблюдателей, участвующим подписантам необходимо шифровать сообщения, которыми они обмениваются.

Алгоритм обмена Диффи-Хеллмана (ECDH) представляет собой довольно простой способ шифрования сообщений при помощи криптографии на эллиптических кривых. Мы уже упоминали об этом в подпункте 5.3, где суммы в выходах Monero сообщаются получателям с использованием общего секрета $rK^v$. Это выглядит вот так:

$$amount_t = b_t \oplus_8 \mathcal{H}_n(\text{“}amount_t, \mathcal{H}_n(rK_B^v, t))$$

Мы с лёгкостью могли бы распространить это на любое сообщение. Сначала сообщение шифруется как последовательность разрядов, а затем разбивается на части, равные по размеру выходу $\mathcal{H}_n$. После этого генерируется случайное число $r \in \mathbb{Z}_l$ и для всех фрагментов сообщения производится обмен Диффи-Хеллмана. При этом используется публичный ключ получателя $K$. Эти зашифрованные фрагменты отправляются вместе с публичным ключом $rG$ предполагаемому получателю, который затем сможет расшифровать сообщение с помощью общего секрета $krG$. Отправители сообщений также должны создать подпись для своего зашифрованного сообщения (или, для простоты, просто хеш зашифрованного сообщения), чтобы получатели могли подтвердить, что сообщения не были подделаны (подпись будет верифицирована только в случае наличия правильного сообщения $\mathfrak{m}$).

Поскольку шифрование не является важным аспектом работы такой криптовалюты, как Monero, мы не считаем необходимым вдаваться в подробности. Любопытные читатели могут ознакомиться с отличным концептуальным обзором [4] или же с техническим описанием популярной схемы шифрования AES, которое приводится в работе [23]. Кроме того, доктором Бернштейном была разработана схема шифрования, известная как ChaCha [37, 100], которая

src/wallet/wallet2.cpp `export_multisig()`

[2] На момент написания этого отчёта нам были известны три варианта реализации схемы multisig. Первая схема предполагает базовый, «ручной» процесс с использованием CLI (командной строки) [114]. Вторая схема представляет собой действительно превосходную систему MMS (Multisig Messaging System), позволяющую безопасно и в практически полностью автоматизированном режиме использовать схему multisig через CLI [115, 116]. Третья версия используется коммерчески доступным кошельком Exa Wallet, код начальной версии которого выложен в репозитории GitHub: `https://github.com/exantech` (это не актуально для текущей версии). Все три варианта основаны на одной и той же фундаментальной версии кодовой базы, что означает существование, по сути, только одного варианта реализации.

используется в основном варианте реализации Monero для шифрования определенной чувствительной информации, связанной с кошельками пользователей (например, образов ключей имеющихся выходов). src/wallet/ringdb.cpp

## 9.2 Агрегирование ключей для адресов

### 9.2.1 Простейший подход

Допустим, N человек хотят создать групповой адрес с мультиподписью, который мы обозначим $(K^{v,grp}, K^{s,grp})$. Средства могут быть направлены по этому адресу точно так же, как и по любому обычному адресу, но, как мы увидим позже, чтобы потратить эти средства всем людям из N будет необходимо сотрудничать, чтобы подписать транзакции.

Так как все участники из N должны иметь возможность видеть средства, полученные на адрес группы, мы можем сообщить всем групповой ключ просмотра $k^{v,grp}$ (см. подпункты 4.1 и 4.2). Чтобы у всех участников были равные права, ключ просмотра может являться суммой компонентов ключа просмотра, которыми смогут безопасно обмениваться все участники. Для участника $e \in \{1, ..., N\}$ его компонентом базового ключа просмотра будет $k_e^{v,base} \in_R \mathbb{Z}_l$, и все участники смогут вычислить групповой приватный ключ просмотра src/multisig/multisig.cpp `generate_multisig_view_secret_key()`

$$k^{v,grp} = \sum_{e=1}^{N} k_e^{v,base}$$

Аналогичным образом групповой ключ траты $K^{s,grp} = k^{s,grp}G$ может являться суммой основных компонентов приватного ключа траты. Однако если кому-то станут известны все компоненты приватного ключа траты, то ему будет известен и общий приватный ключ траты. Добавьте к этому приватный ключ просмотра, и он сможет подписывать транзакции самостоятельно. Это уже не будет мультиподписью. Это будет простая старая подпись.

Вместо этого мы достигаем того же эффекта, если групповой ключ траты представляет собой сумму публичных ключей траты. Допустим, у участников есть базовые публичные ключи траты $K_e^{s,base}$, которыми они безопасно обмениваются. Теперь пусть каждый из них вычислит src/multisig/multisig.cpp `generate_multisig_N_N()`

$$K^{s,grp} = \sum_{e} K_e^{s,base}$$

Очевидно, что это то же самое, что и

$$K^{s,grp} = (\sum_{e} k_e^{s,base}) * G$$

### 9.2.2 Недостатки простейшего подхода

Использование суммы публичных ключей траты интуитивно понятно и кажется простым, но ведёт к появлению нескольких проблем.

### Проверка на агрегирование ключей

Внешний злоумышленник, которому известны все базовые публичные ключи траты $K_e^{s,base}$, может легко проверить имеющийся публичный адрес $(K^v, K^s)$ на предмет агрегирования ключей, вычислив $K^{s,grp} = \sum_e K_e^{s,base}$ и проверив $K^s \stackrel{?}{=} K^{s,grp}$. А это уже связано с более широким требованием, согласно которому агрегированные ключи должны быть неотличимы от обычных ключей, чтобы наблюдатели не могли ничего узнать о действиях пользователей на основе типа публикуемого адреса.[3]

Мы можем решить задачу, создав новые базовые ключи траты для каждого адреса с мультиподписью или замаскировав старые ключи. Первый вариант прост, но может быть связан с определёнными неудобствами.

Второй вариант реализуется следующим образом: при наличии пары старых ключей участника $e$ $(K_e^v, K_e^s)$ с приватными ключами $(k_e^v, k_e^s)$ и случайными масками $\mu_e^v, \mu_e^s$ [4], компоненты его нового базового приватного ключа группового адреса будут следующими:

$$k_e^{v,base} = \mathcal{H}_n(k_e^v, \mu_e^v)$$
$$k_e^{s,base} = \mathcal{H}_n(k_e^s, \mu_e^s)$$

Если участники не хотят, чтобы внешние наблюдатели собирали новые ключи и проверяли их на предмет агрегирования, им придётся безопасно обмениваться своими новыми компонентами ключей.[5]

Если проверка на агрегирование ключей не является для них проблемой, то они могут публиковать свои базовые компоненты $(K_e^{v,base}, K_e^{s,base})$ публичного ключа multisig как обычные адреса. И любая третья сторона на основе этих отдельных адресов сможет вычислить групповой адрес и отправить на него средства, не взаимодействуя с кем-либо из объединившихся получателей [90].

### Аннулирование ключей

Если групповой ключ траты представляет собой сумму публичных ключей, недобросовестный участник, который заранее узнает базовые компоненты базового ключа траты остальных участников, сможет аннулировать их.

[3] Если хотя бы один честный участник использует компоненты, выбранные случайным образом из равномерно распределённых компонентов, то ключи, агрегированные путём простого суммирования, будут неотличимы [122] от обычных ключей.

[4] Случайные маски легко вывести на основе пароля. Например, $\mu^s = \mathcal{H}_n(password)$ и $\mu^v = \mathcal{H}_n(\mu^s)$. Или, как это реализовано в случае Monero, маскируем ключи траты и ключи просмотра строкой, например, $\mu^s, \mu^v =$ “Multisig". Это означает, что Monero поддерживает только один базовый multisig-ключ траты на обычный адрес, хотя на самом деле создание multisig-кошелька приводит к потере доступа пользователем к оригинальному кошельку [114]. Это означает, что Monero поддерживает только один базовый multisig-ключ траты на обычный адрес, хотя на самом деле создание multisig-кошелька приводит к потере доступа пользователем к оригинальному кошельку.

src/multisig/multisig.cpp `get_multisig_blinded_secret_key()`

[5] Как будет указано в подпункте 9.6, агрегация ключей не работает в случае со схемой multisig «M из N», если M < N, из-за наличия общих секретов.

Например, Элис и Боб хотят создать групповой адрес. Элис добросовестно сообщает Бобу свои компоненты ключа $(k_A^{v,base}, K_A^{s,base})$. Боб приватно создаёт свои компоненты ключа $(k_B^{v,base}, K_B^{s,base})$, но не сообщает их Элис сразу. Вместо этого он вычисляет $K_B'^{s,base} = K_B^{s,base} - K_A^{s,base}$ и сообщает Элис $(k_B^{v,base}, K_B'^{s,base})$. Групповой адрес будет следующим:

$$\begin{aligned} K^{v,grp} &= (k_A^{v,base} + k_B^{v,base})G \\ &= k^{v,grp}G \\ K^{s,grp} &= K_A^{s,base} + K_B'^{s,base} \\ &= K_A^{s,base} + (K_B^{s,base} - K_A^{s,base}) \\ &= K_B^{s,base} \end{aligned}$$

В результате получаем групповой адрес $(k^{v,grp}G, K_B^{s,base})$, где Элис известен групповой приватный ключ просмотра, а Бобу известны как приватный ключ просмотра, так и приватный ключ траты! Боб может подписывать транзакции самостоятельно, обманывая тем самым Элис, которая будет уверена в том, что средства, отправленные на адрес, могут быть потрачены только с её разрешения.

Эту проблему можно было бы решить, потребовав перед объединением ключей от каждого из участников создать подпись, подтверждающую, что им известен приватный ключ для их компонента ключа траты [109].[6] Но это неудобно и предполагает наличие уязвимости с точки зрения ошибок реализации. К счастью, в данном случае имеется надёжная альтернатива.

### 9.2.3 Устойчивая агрегация ключей

Простой способ решения проблемы аннулирования ключей состоит во внесении небольших изменений в процесс агрегации ключей (при этом процесс агрегации ключей просмотра остаётся прежним). Допустим, набор из компонентов базовых ключей траты N подписантов $\mathbb{S}^{base} = \{K_1^{s,base}, ..., K_N^{s,base}\}$ упорядочен в соответствии с некоторым соглашением (например, от наименьшего к наибольшему в численном порядке, то есть лексикографически).[7] Надёжный агрегированный ключ траты будет следующим [111][8,9]

$$K^{s,grp} = \sum_e \mathcal{H}_n(T_{agg}, \mathbb{S}^{base}, K_e^{s,base}) K_e^{s,base}$$

Теперь, если Боб попытается аннулировать ключ траты Элис, он столкнется с очень сложной проблемой.

[6] Текущая (и первая) итерация схемы multisig, используемая Monero и реализованная в апреле 2018 года [62] (с интеграцией схемы «M-из-N», реализованной в октябре 2018 года [11]), предполагает этот простой вариант агрегации ключей и требует от пользователей предоставления их компонентов ключа траты. src/wallet/wallet2.cpp `get_multi-sig_info()`

[7] $\mathbb{S}^{base}$ необходимо последовательно упорядочить, чтобы участники могли быть уверены в том, что все они хешируют одно и то же.

[8] В подпункте 3.6 сказано, что хеш-функции должны быть разделены по доменам путём добавления к ним префиксов при помощи тегов, например $T_{agg}$ = “Multisig_Aggregation". Мы оставляем теги для примеров, таких как подписи Шнорра, которые приводятся в следующем разделе.

[9] Важно включать $\mathbb{S}^{base}$ в агрегированные хеши, так как это позволит избежать атак с аннулированием ключей, включающих обобщенное решение Вагнера для «парадокса дней рождения» [137]. [139] [90]

$$\begin{aligned} K^{s,grp} &= \mathcal{H}_n(T_{agg}, \mathbb{S}, K_A^s)K_A^s + \mathcal{H}_n(T_{agg}, \mathbb{S}, K_B'^s)K_B'^s \\ &= \mathcal{H}_n(T_{agg}, \mathbb{S}, K_A^s)K_A^s + \mathcal{H}_n(T_{agg}, \mathbb{S}, K_B'^s)K_B^s - \mathcal{H}_n(T_{agg}, \mathbb{S}, K_B'^s)K_A^s \\ &= [\mathcal{H}_n(T_{agg}, \mathbb{S}, K_A^s) - \mathcal{H}_n(T_{agg}, \mathbb{S}, K_B'^s)]K_A^s + \mathcal{H}_n(T_{agg}, \mathbb{S}, K_B'^s)K_B^s \end{aligned}$$

Мы предлагаем читателю самому представить всю степень разочарования Боба.

Как и в случае с простейшим подходом, любая третья сторона, которой будет известен $\mathbb{S}^{base}$ и соответствующие публичные ключи просмотра, сможет вычислить групповой адрес.

Поскольку участникам не нужно доказывать, что им известны их приватные ключи траты, или взаимодействовать друг с другом, прежде чем подписать транзакцию, наш способ надёжной агрегации ключей соответствует так называемой модели простого публичного ключа, где единственное требование состоит в том, чтобы у каждого потенциального подписанта имелся публичный ключ [90].[10]

#### Функции `premerge` и `merge`

Более формально и для ясности в будущем мы обозначим, что существует операция `premerge (предварительного слияния)`, которая берёт набор базовых ключей $\mathbb{S}^{base}$ и выдаёт набор агрегированных ключей $\mathbb{K}^{agg}$ равного размера, где элемент[11]

$$\mathbb{K}^{agg}[e] = \mathcal{H}_n(T_{agg}, \mathbb{S}^{base}, K_e^{s,base})K_e^{s,base}$$

Агрегированные приватные ключи $k_e^{agg}$ используются в групповых подписях.[12]

Существует ещё одна операция, которая называется `merge (слияние)`. При её выполнении берутся агрегированные ключи, полученные после выполнения операции `premerge`, и создаётся групповой подписывающий ключ (например, в случае Monero это будет ключ траты).

$$K^{grp} = \sum_e \mathbb{K}^{agg}[e]$$

Мы обобщаем эти функции для схем «(N-1) из N» и «M из N» в подпункте 9.6.2 и далее обобщаем их для вложенных мультиподписей в подпункте 9.7.2.

## 9.3 Пороговые подписи, подобные подписям Шнорра

Чтобы мультиподпись работала, требуется определённое количество подписывающих лиц, поэтому существует определённое «пороговое значение» для количества подписывающих лиц,

[10] Как будет указано далее, агрегирование ключей соответствует только простой модели публичного ключа для схем multisig «N из N» и «1 из N».

[11] Примечание: $\mathbb{K}^{agg}[e]$ іявляется членом $e$-нного множества.

[12] Надёжный вариант агрегации ключей ещё не реализован в Monero, но поскольку участники могут хранить и использовать приватный ключ $k_e^{agg}$ (для простейшего варианта агрегации ключей, $k_e^{agg} = k_e^{base}$), обновление Monero с целью использования надёжного варианта агрегации ключей изменит только процесс выполнения операции premerge.

ниже которого подпись не может быть создана. Мультиподпись, создаваемая N участниками, то есть требующая наличия подписи всех N участников и обычно называемая *мультиподписью «N из N»*, будет иметь пороговое значение N. Позже это можно расширить до схемы «M из N» (при $M \leq N$), где N участников создают групповой адрес, но для создания подписи требуется только M человек.

Давайте немного отойдём от Monero. Все схемы подписей в этом документе основаны на общем доказательстве знания с нулевым разглашением Маурера [87], поэтому мы можем продемонстрировать определяющую форму пороговых подписей, используя простую подпись, подобную подписи Шнорра (см. подпункт 2.3.5) [109].

### Подпись

Допустим, есть N пользователей, каждый из которых имеет публичный ключ в наборе $\mathbb{K}^{agg}$, где каждому пользователю $e \in \{1, ..., N\}$ известен приватный ключ $k_e^{agg}$. Их групповым публичным ключом «N из N», который они будут использовать для подписи сообщений, будет $K^{grp}$. Предположим, они хотят вместе подписать сообщение $\mathfrak{m}$. Они могли бы вместе создать базовую подпись, подобную подписи Шнорра, следующим образом

1. Каждый участник $e \in \{1, ..., N\}$ делает следующее:
   (a) выбирает случайный компонент $\alpha_e \in_R \mathbb{Z}_l$,
   (b) вычисляет $\alpha_e G$
   (c) создаёт для него обязательство при помощи $C_e^{\alpha} = \mathcal{H}_n(T_{com}, \alpha_e G)$,
   (d) безопасным образом отправляет $C_e^{\alpha}$ другим участникам.
2. Как только все обязательства $C_e^{\alpha}$ будут собраны, каждый из участников безопасным образом отправляет свои $\alpha_e G$ другим участникам. Они должны верифицировать $C_e^{\alpha} \stackrel{?}{=} \mathcal{H}_n(T_{com}, \alpha_e G)$ для всех остальных участников.
3. Каждый участник вычисляет
$$\alpha G = \sum_e \alpha_e G$$
4. Каждый участник $e \in \{1, ..., N\}$ делает следующее:[13]
   (a) вычисляет запрос $c = \mathcal{H}_n(\mathfrak{m}, [\alpha G])$,
   (b) определяет компонент запроса $r_e = \alpha_e - c * k_e^{agg} \pmod{l}$,
   (c) безопасным образом отправляет re другим участникам.

[13] Как сказано в подпункте 2.3.4, важно не использовать повторно $\alpha_e$ для решения различных запросов $c$. Это означает, что для перезапуска процесса создания мультиподписи, когда ответы уже были отправлены, необходимо начать всё заново с новых значений $\alpha_e$.

5. Каждый участник вычисляет
$$r = \sum_e r_e$$
6. Любой участник может опубликовать подпись $\sigma(\mathfrak{m}) = (c, r)$.

### Верификация

При наличии $K^{grp}$ $\mathfrak{m}$ и $\sigma(\mathfrak{m}) = (c, r)$ выполняем следующее:

1. Вычисляем запрос $c' = \mathcal{H}_n(\mathfrak{m}, [rG + c * K^{grp}])$.
2. Если $c = c'$, подпись является легитимной, за исключением незначительной вероятности.

Мы включили надстрочный индекс $grp$ для ясности, но на самом деле верификатор не будет знать, является ли $K^{grp}$ объединённым ключом, если участник не скажет ему об этом или пока он не узнает компоненты базового или агрегированного ключа.

### Почему это работает

Ответ $r$ является основой этой подписи. Участнику $e$ известны два секрета в $r_e$ ($\alpha_e$ и $k_e^{agg}$), поэтому его приватный ключ $k_e^{agg}$ информационно-теоретически защищён от других участников (при условии, что $\alpha_e$ никогда не будет использован повторно). Более того, верификаторы используют групповой публичный ключ $K^{grp}$, поэтому для построения подписи требуются все ключевые компоненты.

$$\begin{aligned}
rG &= (\sum_e r_e)G \\
&= (\sum_e (\alpha_e - c * k_e^{agg}))G \\
&= (\sum_e \alpha_e)G - c * (\sum_e k_e^{agg})G \\
&= \alpha G - c * K^{grp} \\
\alpha G &= rG + c * K^{grp} \\
\mathcal{H}_n(\mathfrak{m}, [\alpha G]) &= \mathcal{H}_n(\mathfrak{m}, [rG + c * K^{grp}]) \\
c &= c'
\end{aligned}$$

### Дополнительный этап обязательства и раскрытия

Читателя может заинтересовать, откуда взялся этот Этап 2. Без этапа обязательства и раскрытия [111] недобросовестный участник подписания может узнать все $\alpha_e G$ *до того*, как будет вычислен запрос. Это позволит ему в некоторой степени контролировать создаваемый

запрос путём изменения собственного $\alpha_e G$ перед отправкой. Он может использовать компоненты ответа, собранные из нескольких контролируемых подписей, для выведения приватных ключей $k_e^{agg}$ других подписантов за субэкспоненциальное время [53], что представляет собой серьёзную угрозу с точки зрения безопасности. В основе данной угрозы лежит обобщение Вагнера [137] (см. также работу [139], содержащую более содержательное объяснение) парадокса дней рождения [141].[14]

## 9.4 Кольцевые конфиденциальные подписи MLSTAG в Monero

Транзакции Monero, использующие пороговые конфиденциальные кольцевые подписи, предполагают наличие некоторой сложности, поскольку подписывающие ключи MLSTAG (пороговой подписи MLSAG) являются одноразовыми адресами и обязательствами по нулевой сумме (для сумм во входах).

Как говорилось в подпункте 4.2.1, одноразовый адрес, определяющий принадлежность $t$-ного выхода транзакции тому, у кого есть публичный адрес $(K_t^v, K_t^s)$, выглядит следующим образом:

$$K_t^o = \mathcal{H}_n(rK_t^v, t)G + K_t^s = (\mathcal{H}_n(rK_t^v, t) + k_t^s)G$$
$$k_t^o = \mathcal{H}_n(rK_t^v, t) + k_t^s$$

Мы можем обновить наше обозначение выходов, полученных по групповому адресу $(K_t^{v,grp}, K_t^{s,grp})$:

$$K_t^{o,grp} = \mathcal{H}_n(rK_t^{v,grp}, t)G + K_t^{s,grp}$$
$$k_t^{o,grp} = \mathcal{H}_n(rK_t^{v,grp}, t) + k_t^{s,grp}$$

Как и раньше, любой, у кого есть $k_t^{v,grp}$ и $K_t^{s,grp}$, может обнаружить, что $K_t^{o,grp}$ является выходом, принадлежащим его адресу, и расшифровать элемент алгоритма Диффи-Хеллмана для суммы выхода, после чего восстановить соответствующую маску обязательства (см. подпункт 5.3).

Это также означает возможность использования подадресов с мультиподписями (см. подпункт 4.3). Создание multisig-транзакций, использующих средства, полученные на подадрес, требует внесения некоторых простых изменений в алгоритмы, которые упоминаются в сносках.[15]

### 9.4.1 Транзакции `RCTTypeBulletproof2`, построенные по схеме multisig «N из N»

Большинство составляющих multisig-транзакции может быть выполнено тем, кто ее инициировал. И только подписи MLSTAG требуют сотрудничества. Чтобы подготовиться к транзак-

[14] Этап обязательства и раскрытия не используется в текущем варианте реализации мультиподписи Monero, хотя его введение и возможно в будущих версиях. [102]

[15] Подадреса multisig поддерживаются в Monero.

ции типа `RCTTypeBulletproof2`, инициатор должен сделать следующее (см. подпункт 6.2):

1. Сгенерировать приватный ключ транзакции $r \in_R \mathbb{Z}_l$ (см. подпункт 4.2) и вычислить соответствующий публичный ключ $rG$ (или несколько таких ключей, если получатель использует подадреса; см. подпункт 4.3).

2. Определить, какие входы будут потрачены ($j \in \{1, ..., m\}$ собственные выходы с одноразовыми адресами $K_j^{o,grp}$ и суммами $a_1, ..., a_m$), и получателей, которые получат средства ($t \in \{0, ..., p-1\}$ из новых выходов с суммами $b_0, ..., b_{p-1}$ и одноразовые адреса $K_t^o$). Сюда входит комиссия $f$, получаемая майнером, и его обязательство $fH$. Затем определить набор ложных участников кольца для каждого входа.

3. Зашифровать сумму каждого $amount_t$ (см. подпункт 5.3) и вычислить обязательства по выходу $C_t^b$.

4. Выбрать для каждого входа $j \in \{1, ..., m-1\}$ компоненты маски обязательства по псевдо-выходу $x'_j \in_R \mathbb{Z}_l$ и вычислить $m$-ную маску следующим образом (см. подпункт 5.4)
$$x'_m = \sum_t y_t - \sum_{j=1}^{m-1} x'_j$$
Вычислить обязательства по псевдо-выходу $C'^a_j$.

5. Создать агрегированное доказательство диапазона Bulletproof для всех выходов. См. подпункт 5.5.

6. Подготовиться к созданию подписей MLSTAG, сгенерировав начальные компоненты $\alpha_j^z \in_R \mathbb{Z}_l$ для обязательств по нулевой сумме и вычислив $\alpha_j^z G$.[16]

После этого инициатор безопасным образом отправляет всю эту информацию другим участникам. Теперь группа подписантов готова к созданию подписей входов при помощи своих приватных ключей $k_e^{s,agg}$ и обязательств по нулевой сумме $C^a_{\pi,j} - C'^a_{\pi,j} = z_j G$.

### MLSTAG RingCT

Сначала создаются образы групповых ключей для всех входов $j \in \{1, ..., m\}$ с одноразовыми адресами $K_{\pi,j}^{o,grp}$.[17]

src/wallet/wallet2.cpp `sign_multisig_tx()`

1. Для каждого входа $j$ каждый участник $e$ выполняет следующее:

[16] Здесь нет никакой необходимости в этапе обязательства и раскрытия, поскольку обязательства по нулевой сумме известны всем подписантам.

[17] Если $K_{\pi,j}^{o,grp}$ построена на основе multisig-подадреса с индексом $i$, то (см. подпункт 4.3) приватный ключ будет составным:
$$k_{\pi,j}^{o,grp} = \mathcal{H}_n(k^{v,grp} r_{u_j} K^{s,grp,i}, u_j) + \sum_e k_e^{s,agg} + \mathcal{H}_n(k^{v,grp}, i)$$

(a) вычисляет частичный образ ключа $\tilde{K}^o_{j,e} = k^{s,agg}_e \mathcal{H}_p(K^{o,grp}_{\pi,j})$,

(b) безопасным образом отправляет $\tilde{K}^o_{j,e}$ другим участникам.

2. Теперь каждый участник, используя $u_j$ в качестве индекса выход транзакции, может вычислить, куда именно $K^{o,grp}_{\pi,j}$ был отправлен на multisig-адрес,[18]

$$\tilde{K}^{o,grp}_j = \mathcal{H}_n(k^{v,grp} rG, u_j)\mathcal{H}_p(K^{o,grp}_{\pi,j}) + \sum_e \tilde{K}^o_{j,e}$$

Затем для каждого входа $j$ создаётся подпись MLSTAG.

1. Каждый участник $e$ делает следующее:

(a) генерирует начальные компоненты $\alpha_{j,e} \in_R \mathbb{Z}_l$ и вычисляет $\alpha_{j,e}G$ $\alpha_{j,e}\mathcal{H}_p(K^{o,grp}_{\pi,j})$, src/wallet/wallet2.cpp `get_multisig_kLRki()`

(b) генерирует случайные компоненты $r_{i,j,e}$ и $i \in \{1, ..., v+1\}$ для $r^z_{i,j,e}$, за исключением случаев, когда $i = \pi$

(c) вычисляет обязательство

$$C^\alpha_{j,e} = \mathcal{H}_n(T_{com}, \alpha_{j,e}G, \alpha_{j,e}\mathcal{H}_p(K^{o,grp}_{\pi,j}), r_{1,j,e}, ..., r_{v+1,j,e}, r^z_{1,j,e}, ..., r^z_{v+1,j,e})$$

(d) безопасным образом отправляет $C^\alpha_{j,e}$ другим участникам.

2. После получения всех $C^\alpha_{j,e}$ от других участников отправляется все $\alpha_{j,e}G$, $\alpha_{j,e}\mathcal{H}_p(K^{o,grp}_{\pi,j})$, $r_{i,j,e}$, и $r^z_{i,j,e}$ и проверяет легитимность исходных обязательств каждого участника.

3. Каждый участник может вычислить все

$$\alpha_j G = \sum_e \alpha_{j,e} G$$

$$\alpha_j \mathcal{H}_p(K^{o,grp}_{\pi,j}) = \sum_e \alpha_{j,e} \mathcal{H}_p(K^{o,grp}_{\pi,j})$$

$$r_{i,j} = \sum_e r_{i,j,e}$$

$$r^z_{i,j} = \sum_e r^z_{i,j,e}$$

4. Каждый участник может построить цикл подписи (см. подпункт 3.5).

5. Чтобы завершить подпись, каждый участник $e$ делает следующее:

(a) определяет $r_{\pi,j,e} = \alpha_{j,e} - c_\pi k^{s,agg}_e \pmod l$,

(b) безопасным образом отправляет $r_{\pi,j,e}$ другим участникам.

[18] Если одноразовый адрес соответствует multisig-подадресу с индексом $i$, следует добавить

$$\tilde{K}^{o,grp}_j = ... + \mathcal{H}_n(k^{v,grp}, i)\mathcal{H}_p(K^{o,grp}_{\pi,j})$$

src/cryptonote_basic/cryptonote_format_utils.cpp `generate_key_image_helper_precomp()`

6. Каждый может вычислить (учитывая, что $\alpha^z_{j,e}$ была создана инициатором транзакции)[19] src/ringct/rctSigs.cpp `signMultisig()`

$$r_{\pi,j} = \sum_e r_{\pi,j,e} - c_\pi * \mathcal{H}_n(k^{v,grp} rG, u_j)$$
$$r^z_{\pi,j} = \alpha^z_{j,e} - c_\pi z_j \pmod l$$

Подпись для входа $j$ будет $\sigma_j(\mathfrak{m}) = (c_1, r_{1,j}, r^z_{1,j}, ..., r_{v+1,j}, r^z_{v+1,j})$ с $\tilde{K}^{o,grp}_j$.

Поскольку в случае с Monero сообщение $\mathfrak{m}$ и запрос $c_\pi$ зависят от всех других частей транзакции, любой участник, который попытается обмануть остальных участников, отправив им неправильное значение в какой-либо момент реализации процесса, сделает так, что подпись просто будет недействительной. Ответ $r_{\pi,j}$ полезен только для $\mathfrak{m}$ и $c_\pi$, для которых он и определён.

### 9.4.2 Упрощённый вариант обмена данными

Для создания транзакции Monero с мультиподписью требуется пройти множество этапов. Мы можем реорганизовать и упростить некоторые из них, чтобы взаимодействие подписантов происходило в два этапа в совокупности всего за пять раундов.

1. *Агрегация ключей для публичного nultisig-адреса*: любой, у кого имеется набор публичных адресов, может выполнить с ними операцию `premerge`, а затем объединить их (операция `merge`) в адрес «N из N», но ни один из участников не будет знать групповой ключ просмотра до тех пор, пока не узнает все компоненты. Поэтому создание группы начинается с безопасного обмена $k^v_e$ и $K^{s,base}_e$ между участниками. Любой участник может выполнить операции `premerge` и `merge` и опубликовать $(K^{v,grp}, K^{s,grp})$, что позволит группе получать средства на групповой адрес. Агрегирование по схеме «M из N» требует прохождения дополнительных этапов, описанных в подпункте 9.6. src/wallet/wallet2.cpp `pack_multisignature_keys()`
2. *Транзакции*:
   (a) Какой-то участник или подгруппа участников (инициатор) решает создать транзакцию. Он выбирает $m$ входов с одноразовыми адресами $K^{o,grp}_j$ и обязательствами по сумме $C^a_j$, $m$ наборов из $v$ дополнительных одноразовых адресов и обязательства, которые будут использоваться в кольце в качестве ложных, выбирает $p$ получателей выходов с публичными адресами $(K^v_t, K^s_t)$ и суммами $b_t$, которые будут им отправлены, определяет размер комиссии $f$ за транзакцию, выбирает приватный ключ транзакции $r$,[20] генерирует маски обязательств по псевдовыходам $x'_j$, где $j \neq m$, создаёт элемент ECDH $amount_t$ для каждого выхода, создаёт агрегированное src/wallet/wallet2.cpp `transfer_selected_rct()`

[19] Если одноразовый адрес $K^{o,grp}_{\pi,j}$ соответствует multisig-подадресу с индексом $i$, следует добавить

$$r_{\pi,j} = ... - c_\pi * \mathcal{H}_n(k^{v,grp}, i)$$

src/cryptonote_basic/cryptonote_format_utils.cpp `generate_key_image_helper_precomp()`

[20] Или же приватные ключи транзакции $r_t$ при отправке хотя бы на один подадрес.

доказательство диапазона и генерирует средства для открытия подписи $\alpha_j^z$ для всех обязательств по нулевой сумме входов и случайные скалярные величины $r_{i,j}$ и $r_{i,j}^z$ с $i \neq \pi_j$.[21] Также он подготавливается к участию в следующем раунде обмена данными.

Инициатор безопасным образом отправляет всю эту информацию другим участникам.[22] Другие участники могут сообщить ему о своём согласии, отправив свою часть данных для следующего раунда, или же могут согласовывать изменения. src/wallet/wallet2.cpp `save_multisig_tx()`

(b) Каждый участник выбирает свои открывающие подпись (подписи) MLSTAG компоненты, даёт по ним обязательства, вычисляет частичные образы ключей и безопасным образом отправляет эти обязательства и частичные образы другим участникам.

Подпись (подписи) MLSTAG: образ ключа $\tilde{K}_{j,e}^o$, степень случайности подписи $\alpha_{j,e}G$ и $\alpha_{j,e}\mathcal{H}_p(K_{\pi,j}^{o,grp})$. Частичные образы ключей необязательно должны присутствовать в данных, по которым даются обязательства, поскольку их нельзя использовать для извлечения приватных ключей подписантов. Они также позволяют посмотреть, какие из имеющихся выходов были потрачены. Поэтому в целях сохранения модульной структуры их следует обрабатывать отдельно.

(c) После получения всех обязательств подписи каждый участник безопасным образом отправляет информацию, по которой были даны обязательства, другим участникам.

(d) ЕКаждый участник закрывает свою часть подписи (подписей) MLSTAG, безопасным образом отправляя все $r_{\pi_j,j,e}$ другим участникам.[23]

Если процесс был реализован успешно, все участники могут закончить создание транзакции и транслировать её самостоятельно. Транзакции, созданные участниками, использующими схему multisig, неотличимы от транзакций, созданных отдельными пользователями.

## 9.5 Пересчёт образов ключей

Если кто-то потеряет свои записи и захочет вычислить баланс на своём адресе (полученные средства минус потраченные средства), ему будет необходимо свериться с данными блокчейна.

[21] Следует отметить, что мы упрощаем процесс подписания, допуская, что инициатор будет генерировать случайные скалярные величины $r_{i,j}$ и $r_{i,j}^z$ вместо того, чтобы каждый из подписантов генерировал компоненты, которые в конечном счёте будут суммироваться.

[22] От него не требуется отправлять суммы в выходах $b_t$ напрямую, так как они могут быть вычислены на основе $amount_t$. Monero использует разумный подход к созданию частичной транзакции, заполненной информацией, выбранной инициатором, и к её отправке другим подписантам вместе со списком соответствующей информации, такой как приватные ключи транзакции, адреса назначения, реальные входы и т. д.

[23] Крайне важно, чтобы при каждой попытке подписания со стороны подписанта использовались уникальные $\alpha_{j,e}$, так как это позволяет избежать раскрытия приватного ключа траты этой стороны другим подписантам (см. подпункт 2.3.4) [111]. Кошельки должны строго соблюдать это правило и удалять $\alpha_{j,e}$ всякий раз, когда ответ, в котором используется этот компонент, передаётся куда-либо за пределы кошелька. src/wallet/wallet2.cpp `save_multisig_tx()`

Ключи просмотра позволяют узнать только то, сколько было получено. Поэтому пользователям необходимо вычислить образы ключей всех принадлежащих им выходов и сравнить их с образами ключей, которые хранятся в блокчейне, чтобы узнать, были ли они потрачены. Поскольку участники группового адреса не имеют возможности вычислить образы ключей самостоятельно, им требуется заручиться поддержкой других участников.

Вычисление образов ключей на основе простой суммы компонентов может быть безуспешным, если недобросовестные участники передадут ложные ключи.[24] При наличии полученного выхода с одноразовым адресом $K^{o,grp}$ группа может создать простое «связываемое» доказательство, подобное доказательству Шнорра (в основном bLSTAG с одним ключом, см. подпункты 3.1 и 3.4), чтобы доказать, что образ $\tilde{K}^{o,grp}$ ключа является легитимным, не раскрывая при этом приватных компонентов ключа траты и без необходимости в каком-либо доверии друг к другу.

### Доказательство

1. Каждый участник $e$ делает следующее:
   (a) вычисляет $\tilde{K}_e^o = k_e^{s,agg}\mathcal{H}_p(K^{o,grp})$,
   (b) генерирует начальный компонент $\alpha_e \in_R \mathbb{Z}_l$ и вычисляет $\alpha_e G$ и $\alpha_e\mathcal{H}_p(K^{o,grp})$,
   (c) даёт обязательства по данным, используя $C_e^\alpha = \mathcal{H}_n(T_{com}, \alpha_e G, \alpha_e\mathcal{H}_p(K^{o,grp}))$,
   (d) безопасным образом отправляет $C_e^\alpha$ и $\tilde{K}_e^o$ другим участникам.
2. После получения всех $C_e^\alpha$ каждый участник отправляет информацию, по которой были даны обязательства, и проверяет, являются ли обязательства других участников легитимными.
3. Каждый участник может вычислить:[25]
$$\tilde{K}^{o,grp} = \mathcal{H}_n(k^{v,grp}rG, u)\mathcal{H}_p(K^{o,grp}) + \sum_e \tilde{K}_e^o$$
$$\alpha G = \sum_e \alpha_e G$$
$$\alpha\mathcal{H}_p(K^{o,grp}) = \sum_e \alpha_e\mathcal{H}_p(K^{o,grp})$$
4. Каждый участник вычисляет запрос[26]
$$c = \mathcal{H}_n([\alpha G], [\alpha\mathcal{H}_p(K^{o,grp})])$$

[24] В настоящее время Monero использует простую сумму.

[25] Если одноразовый адрес соответствует multisig-подадресу с индексом $i$, следует добавить
$$\tilde{K}^{o,grp} = ... + \mathcal{H}_n(k^{v,grp}, i)\mathcal{H}_p(K^{o,grp})$$

src/wallet/wallet2.cpp `export_multisig()`

[26] Данное доказательство должно использовать разделение по доменам и добавление префикса к ключам, которые мы опускаем для краткости.

5. Каждый участник делает следующее:
   (a) определяет $r_e = \alpha_e - c * k_e^{s,agg} \pmod{l}$,
   (b) безопасным образом отправляет $r_e$ другим участникам.
6. Каждый участник может вычислить[27]
$$r^{resp} = \sum_e r_e - c * \mathcal{H}_n(k^{v,grp} rG, u)$$

Доказательство $(c, r^{resp})$ with $\tilde{K}^{o,grp}$.

### Верификация

1. Проверить $l\tilde{K}^{o,grp} \stackrel{?}{=} 0$.
2. Вычислить $c' = \mathcal{H}_n([r^{resp}G + c * K^{o,grp}], [r^{resp}\mathcal{H}_p(K^{o,grp}) + c * \tilde{K}^{o,grp}])$.
3. Если $c = c'$, то образ ключа $\tilde{K}^{o,grp}$ соответствует одноразовому адресу $K^{o,grp}$ (за исключением ничтожной вероятности).

## 9.6 Меньшие пороговые значения

В начале этой главы говорилось об эскроу-сервисах, использующих схему multisig «2 из 2» для разделения полномочий по созданию подписи между пользователем и компанией, отвечающей за обеспечение безопасности. Такая схема не идеальна, потому что, если компания подвергнется взлому или откажется от сотрудничества, ваши средства могут «зависнуть».

Такого варианта развития событий можно избежать, если использовать multisig-адрес по схеме «2 из 3», которая предполагает наличие трёх участников, но для подписания транзакций при этом нужны только двое из них. Эскроу-сервис предоставляет один ключ, а пользователи - два ключа. Пользователь может хранить один ключ в безопасном месте (например, в сейфе), а другой использовать для повседневных покупок. Если с эскроу-сервисом что-либо произойдёт, пользователь сможет использовать ключ безопасности и ключ для повседневного использования для вывода своих средств.

Мультиподписи с пороговым значением ниже N имеют широкий спектр применения.

[27] Если одноразовый адрес $K^{o,grp}$ соответствует multisig-подадресу с индексом $i$, следует добавить
$$r^{resp} = ... - c * \mathcal{H}_n(k^{v,grp}, i)$$

### 9.6.1 Агрегирование ключей по схеме «1 из N»

Предположим, группа людей хочет создать multisig-ключ $K^{grp}$, который могли бы использовать для подписания все участники. Решение тривиально: пусть всем будет известен приватный ключ $k^{grp}$. Это можно сделать тремя способами.

1. Один участник или подгруппа участников выбирает ключ и безопасным образом отправляет его всем остальным.
2. Все участники вычисляют компоненты приватного ключа и безопасным образом отправляют их, используя простую сумму в качестве объединённого ключа.[28]
3. Участники расширяют мультиподпись «M из N» до «1 из N». Это поможет, если злоумышленник имеет доступ к сообщениям группы.

В этом случае при использовании Monero все будут знать приватные ключи $(k^{v,grp,1\text{xN}}, k^{s,grp,1\text{xN}})$. До этого раздела все групповые ключи использовали схему «N из N», но теперь мы используем верхний индекс 1xN для обозначения ключей, связанных с подписью «1 из N»

### 9.6.2 Агрегирование ключей по схеме «(N-1) из N»

В случае с групповым ключом, созданным по схеме «(N-1) из N», таким как «2 из 3» или «4 из 5», любая группа, состоящая из (N-1) участников, может подписать транзакцию. Мы достигаем этого при помощи общих секретов Диффи-Хеллмана. Допустим, есть участники $e \in \{1, ..., N\}$ с базовыми публичными ключами $K_e^{base}$, которые известны им всем.

Каждый участник $e$ вычисляет $w \in \{1, ..., N\}$, но $w \neq e$

src/multisig/multisig.cpp `generate_multisig_N1_N()`

$$k_{e,w}^{sh,(\text{N-1})\text{xN}} = \mathcal{H}_n(k_e^{base} K_w^{base})$$

Затем он вычисляет все $K_{e,w}^{sh,(\text{N-1})\text{xN}} = k_{e,w}^{sh,(\text{N-1})\text{xN}} G$ и безопасным образом отправляет результат другим участникам. Теперь мы используем верхний индекс $sh$ для обозначения ключей, являющихся общими для подгруппы участников.

У каждого участника будет в наличии (N-1) общих компонентов приватного ключа, соответствующих каждому компоненту других участников, что составит N*(N-1) ключей в совокупности между всеми. В соответствии с алгоритмом Диффи-Хеллмана все ключи используются двумя сторонами-участниками, поэтому на самом деле в наличии будет [N*(N-1)]/2 уникальных ключей. Эти уникальные ключи составляют набор $\mathbb{S}^{(\text{N-1})\text{xN}}$.

[28] Обратите внимание, что аннулирование ключа в данном случае довольно бессмысленно, потому что каждому участнику известен полный приватный ключ.

### Обобщение функций `premerge` и `merge`

Здесь мы обновляем определение `premerge` из подпункта 9.2.3. В данном случае входом будет набор $\mathbb{S}^{\text{MxN}}$, где $M$ является «пороговым значением», для которого были подготовлены ключи из набора. Если $M = N$, $\mathbb{S}^{\text{NxN}} = \mathbb{S}^{base}$, а если $M < N$, значит, набор содержит общие ключи. Выходом является $\mathbb{K}^{agg,\text{MxN}}$.

Компоненты ключей $[\text{N*(N-1)}]/2$ в $\mathbb{K}^{agg,\text{(N-1)xN}}$ можно отправить для слияния (проведения операции `merge`), и получить в результате $K^{grp,\text{(N-1)xN}}$. Важно отметить, что все компоненты приватного ключа $[\text{N*(N-1)}]/2$ могут быть собраны только при наличии (N-1) участников, поскольку каждый участник использует один общий секрет Диффи-Хеллмана с $N$-ым участником.

### Пример схемы «2 из 3»

Предположим, есть три человека, у которых есть публичные ключи $\{K_1^{base}, K_2^{base}, K_3^{base}\}$, и каждому из них известен приватный ключ. При этом они хотят создать multisig-ключ по схеме «2 из 3». После выполнения алгоритма Диффи-Хеллмана и обмена публичными ключами каждому из них будет известно следующее:

1. Участник 1: $k_{1,2}^{sh,\text{2x3}}$, $k_{1,3}^{sh,\text{2x3}}$, $K_{2,3}^{sh,\text{2x3}}$
2. Участник 2: $k_{2,1}^{sh,\text{2x3}}$, $k_{2,3}^{sh,\text{2x3}}$, $K_{1,3}^{sh,\text{2x3}}$
3. Участник 3: $k_{3,1}^{sh,\text{2x3}}$, $k_{3,2}^{sh,\text{2x3}}$, $K_{1,2}^{sh,\text{2x3}}$

Где $k_{1,2}^{sh,\text{2x3}} = k_{2,1}^{sh,\text{2x3}}$, и так далее. Набор $\mathbb{S}^{\text{2x3}} = \{K_{1,2}^{sh,\text{2x3}}, K_{1,3}^{sh,\text{2x3}}, K_{2,3}^{sh,\text{2x3}}\}$.

В результате выполнения операций `premerge` и `merge` создаётся групповой ключ:[29]

$$\begin{aligned} K^{grp,\text{2x3}} =& \mathcal{H}_n(T_{agg}, \mathbb{S}^{\text{2x3}}, K_{1,2}^{sh,\text{2x3}}) K_{1,2}^{sh,\text{2x3}} + \\ & \mathcal{H}_n(T_{agg}, \mathbb{S}^{\text{2x3}}, K_{1,3}^{sh,\text{2x3}}) K_{1,3}^{sh,\text{2x3}} + \\ & \mathcal{H}_n(T_{agg}, \mathbb{S}^{\text{2x3}}, K_{2,3}^{sh,\text{2x3}}) K_{2,3}^{sh,\text{2x3}} \end{aligned}$$

Теперь предположим, что участники 1 и 2 хотят подписать сообщение $\mathfrak{m}$. Для демонстрации мы будем использовать базовую подпись, подобную подписи Шнорра.

1. Каждый участник $e \in \{1, 2\}$ делает следующее:
   (a) выбирает случайный компонент $\alpha_e \in_R \mathbb{Z}_l$,
   (b) вычисляет $\alpha_e G$,

[29] Поскольку объединённый ключ состоит из общих секретов, наблюдатель, которому известны только исходные базовые публичные ключи, не сможет объединить их (см. подпункт 9.2.2) и идентифицировать участников, создавших объединённый ключ.

   (c) создаёт обязательство при помощи $C_e^\alpha = \mathcal{H}_n(T_{com}, \alpha_e G)$,

   (d) безопасным образом отправляет $C_e^\alpha$ другим участникам.

2. После получения всех $C_e^\alpha$ каждый участник отправляет $\alpha_e G$ и проверяет, являются ли другие обязательства легитимными.

3. Каждый участник вычисляет
$$\alpha G = \sum_e \alpha_e G$$
$$c = \mathcal{H}_n(\mathfrak{m}, [\alpha G])$$

4. Участник 1 делает следующее:

   (a) вычисляет $r_1 = \alpha_1 - c * [k_{1,3}^{agg,2\text{x}3} + k_{1,2}^{agg,2\text{x}3}]$,

   (b) безопасным образом отправляет $r_1$ участнику 2.

5. Участник 2 делает следующее:

   (a) вычисляет $r_2 = \alpha_2 - c * k_{2,3}^{agg,2\text{x}3}$,

   (b) безопасным образом отправляет $r_2$ участнику 1.

6. Каждый участник вычисляет
$$r = \sum_e r_e$$

7. Любой из участников может опубликовать подпись $\sigma(\mathfrak{m}) = (c, r)$.

Единственным изменением в случае с пороговыми подписями с пороговым значением ниже N является то, как путём определения $r_{\pi,e}$ «замыкается цикл» (в случае использования кольцевых подписей с секретным индексом $\pi$). Каждый участник должен включить свой общий секрет, соответствующий «отсутствующему участнику», но, поскольку все другие общие секреты дублируются, здесь есть уловка. При наличии набора исходных ключей всех участников $\mathbb{S}^{base}$ только *первый участник*, указываемый в $\mathbb{S}^{base}$ по индексу, вместе с копией общего секрета использует его для вычисления своего $r_{\pi,e}$.[30,31]

В предыдущем примере участник 1 вычисляет

[30] На практике это означает, что инициатор должен определить, какой поднабор из M подписантов будет подписывать данное сообщение. Если выяснится, что O подписантов желает подписать его при O $\geq$ M, инициатор сможет организовать множество одновременных попыток подписания поднабором из M участников в пределах O, что повысить шансы на успех. Monero использует этот подход. Также (что известно посвящённым) *исходный* список адресов, по которым будут направлены выходы, созданный инициатором, перетасовывается случайным образом, а затем этот «случайный» список используется при всех одновременных попытках подписания всеми остальными подписантами (это связано с флагом сокрытия `shuffle_outs`). src/wallet/wallet2.cpp `transfer_selected_rct()`

[31] В настоящее время Monero использует метод кругового подписания, когда инициатор подписывает транзакцию, используя все свои приватные ключи, передаёт частично подписанную транзакцию другому подписанту, который также подписывает *её* всеми своими приватными ключами (которые ещё не использовались для подписания), который тоже передаёт её другому подписанту, и так далее до тех пор, пока последний подписант либо не опубликует транзакцию, либо не отправит её другим подписантам, чтобы те её опубликовали. src/wallet/wallet2.cpp `sign_multisig_tx()`

$$r_1 = \alpha_1 - c * [k_{1,3}^{agg,2x3} + k_{1,2}^{agg,2x3}]$$

в то время как участник 2 вычисляет только

$$r_2 = \alpha_2 - c * k_{2,3}^{agg,2x3}$$

Тот же принцип применяется к вычислению образа группового ключа в multisig-транзакциях Monero с пороговым значением ниже N.

### 9.6.3 Агрегирование ключей по схеме «M из N»

Схему «M из N» можно понять, немного изменив наше представление о схеме «(N-1) из N». В случае со схемой «(N-1) из N» каждый общий секрет между двумя публичными ключами, такой как $K_1^{base}$ и $K_2^{base}$, содержит два приватных ключа, $k_1^{base}k_2^{base}G$. Это секрет, поскольку только участник 1 может вычислить $k_1^{base}K_2^{base}$, и только участник 2 может вычислить $k_2^{base}K_1^{base}$.

А что, если бы существовал третий участник $K_3^{base}$, общие секреты $k_1^{base}k_2^{base}G$, $k_1^{base}k_3^{base}G$ и $k_2^{base}k_3^{base}G$, а участники обменивались бы публичными ключами между собой (что более не делало бы их секретом)? Каждый из участников добавил приватный ключ к двум публичным ключам. Теперь, скажем, они создают общий секрет с использованием третьего публичного ключа.

Участник 1 вычисляет общий секрет $k_1^{base} * (k_2^{base}k_3^{base}G)$, участник 2 вычисляет общий секрет $k_2^{base} * (k_1^{base}k_3^{base}G)$, а участник 3 вычисляет $k_3^{base} * (k_1^{base}k_2^{base}G)$. Теперь всем им известен $k_1^{base}k_2^{base}k_3^{base}G$, что является трёхсторонним общим секретом (при условии, что никто не опубликует его).

src/multisig/multisig.cpp `generate_multisig_derivations()`

Группа могла бы использовать $k^{sh,1x3} = \mathcal{H}_n(k_1^{base}k_2^{base}k_3^{base}G)$ в качестве общего приватного ключа и опубликовать

$$K^{grp,1x3} = \mathcal{H}_n(T_{agg}, \mathbb{S}^{1x3}, K^{sh,1x3})K^{sh,1x3}$$

как multisig-адрес, построенный по схеме «1 из 3».

В случае со схемой multisig «3 из 3» у каждого участника имеется секрет, в мультиподписи, построенной по схеме «2 из 3», каждая группа, состоящая из 2 участников, имеет общий секрет, а в случае со схемой «1 из 3» общий секрет имеется у каждой группы из трёх участников.

Теперь мы можем обобщить схему «M из N»: у каждой возможной группы, состоящей из (N-M + 1) участников, будет иметься общий секрет [109]. В случае отсутствия (N-M) участников все их общие секреты принадлежат, по крайней мере, одному из M оставшихся участников, которые могут совместно подписать транзакцию при помощи группового ключа.

#### Алгоритм построения мультиподписи по схеме «M из N»

При наличии участников $e \in \{1, ..., N\}$ с исходными приватными ключами $k_1^{base}, ..., k_N^{base}$, желающими создать объединённый ключ по схеме «M ≤ N» (где M ≥ 1 и N ≥ 2), мы можем использовать интерактивный алгоритм.

src/wallet/wallet2.cpp `exchange_multisig_keys()`

Для обозначения всех *уникальных* публичных ключей на этапе $\mathbb{S}_s$ используем $s \in \{0, ..., (N - M)\}$. Окончательный набор $\mathbb{S}_{N-M}$ упорядочивается в соответствии с соглашением о распределении (например, от наименьшего к большему численно, то есть лексикографически). Данные обозначения используются для удобства, а $\mathbb{S}_s$ означает то же, что обозначало $\mathbb{S}^{\text{(N-s)xN}}$ в предыдущих подпунктах.

Для обозначения набора публичных ключей, созданных каждым участником на этапе $s$ алгоритма, используем $\mathbb{S}^K_{s,e}$. В начале $\mathbb{S}^K_{0,e} = \{K^{base}_e\}$.

В конце набор $\mathbb{S}^k_e$ будет содержать приватные ключи агрегации каждого участника.

1. Каждый участник $e$ безопасным образом отправляет свой исходный набор публичных ключей $\mathbb{S}^K_{0,e} = \{K^{base}_e\}$ другим участникам.
2. Каждый участник строит $\mathbb{S}_0$, собирая все $\mathbb{S}^K_{0,e}$ и удаляя дубликаты.
3. На этапе $s \in \{1, ..., (N - M)\}$ (пропускаем, если M = N)
   (a) Каждый участник $e$ делает следующее:
      i. Для каждого элемента $g_{s-1}$ $\mathbb{S}_{s-1} \notin \mathbb{S}^K_{s-1,e}$ вычисляет новый общий секрет
      $$k^{base}_e * \mathbb{S}_{s-1}[g_{s-1}]$$
      ii. Использует все новые общие секреты в $\mathbb{S}^K_{s,e}$.
      iii. Если $s = (N - M)$, вычисляет общий приватный ключ для каждого элемента $x$ в $\mathbb{S}^K_{N-M,e}$
      $$\mathbb{S}^k_e[x] = \mathcal{H}_n(\mathbb{S}^K_{N-M,e}[x])$$
      и переписывает публичный ключ, задав $\mathbb{S}^K_{N-M,e}[x] = \mathbb{S}^k_e[x] * G$.
      iv. Отправляет $\mathbb{S}^K_{s,e}$ остальным участникам.
   (b) Каждый участник строит $\mathbb{S}_s$ путём сбора всех $\mathbb{S}^K_{s,e}$ и удаления дубликатов.[32]
4. Каждый участник распределяет $\mathbb{S}_{N-M}$ в соответствии с соглашением.
5. Функция `premerge` использует $\mathbb{S}_{(N-M)}$ в качестве вводных данных и каждый ключ агрегации для $g \in \{1, ..., (\text{size of } \mathbb{S}_{N-M})\}$,
$$\mathbb{K}^{agg,\text{MxN}}[g] = \mathcal{H}_n(T_{agg}, \mathbb{S}_{(N-M)}, \mathbb{S}_{(N-M)}[g]) * \mathbb{S}_{(N-M)}[g]$$
6. Функция `merge` использует $\mathbb{K}^{agg,\text{MxN}}$ в качестве вводных данных, а групповой ключ будет следующим
$$K^{grp,\text{MxN}} = \sum_{g=1}^{\text{size of } \mathbb{S}_{N-M}} \mathbb{K}^{agg,\text{MxN}}[g]$$

[32] Участники должны отслеживать, какие будут ключи и у кого они окажутся на последнем этапе ($s = N-M$), что облегчит совместное подписание, так как только первый участник в $\mathbb{S}_0$ с определённым приватным ключом использует его для подписи. См. подпункт 9.6.2.

7. Каждый участник $e$ переписывает каждый элемент $x$ в $\mathbb{S}_e^k$ при помощи своего приватного ключа агрегации.

$$\mathbb{S}_e^k[x] = \mathcal{H}_n(T_{agg}, \mathbb{S}_{(N-M)}, \mathbb{S}_e^k[x]G) * \mathbb{S}_e^k[x]$$

Примечание: если пользователи хотят иметь неравные полномочия при создании мультиподписи, например 2 «голоса» при использовании схемы «3 из 4», им следует использовать несколько компонентов стартового ключа, а не повторно использовать один и тот же.

## 9.7 Семейства ключей

До этого момента мы рассматривали ключ агрегации простой группы подписантов. Например, Элис, Боб и Кэрол выдают компоненты ключа для multisig-адреса, создаваемого по схеме «2 из 3».

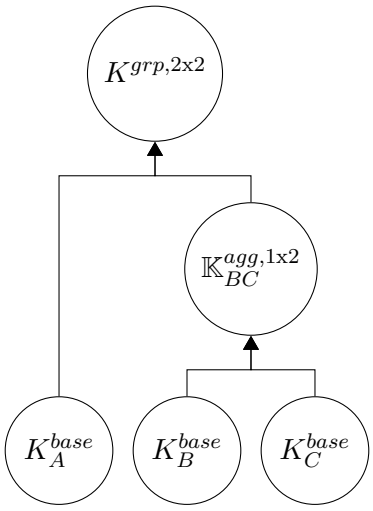
Теперь представьте, что Элис хочет подписать все транзакции с этого адреса, но не хочет, чтобы Боб и Кэрол подписывались без её участия. Другими словами, вариант (Элис + Боб) или (Элис + Кэрол) приемлем, но не вариант (Боб + Кэрол).

Мы можем реализовать этот сценарий, используя два уровня агрегирования ключей. На первом уровне происходит агрегирование по схеме multisig «1 из 2» $\mathbb{K}_{BC}^{agg,1\text{x}2}$ между Бобом и Кэрол, а затем по схеме «2 из 2» строится групповой ключ $K^{grp,2\text{x}2}$ между Элис и $\mathbb{K}_{BC}^{agg,1\text{x}2}$. Как правило, это multisig-адрес, построенный по схеме «(2 из ([1 из 1] и [1 из 2]))».

Это означает, что структуры предоставления доступа к праву подписи могут быть довольно открытыми.

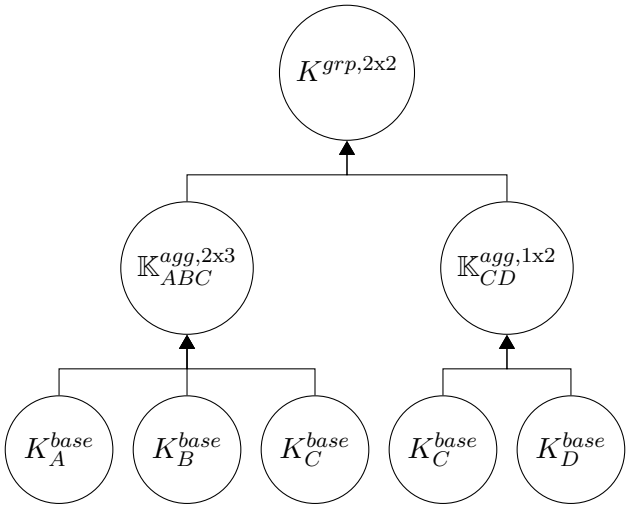
### 9.7.1 Деревья семейств

Мы можем представить диаграмму multisig-адреса, построенного по схеме «(2 из ([1 из 1] и [1 из 2]))», следующим образом:



Ключи $K_A^{base}, K_B^{base}, K_C^{base}$ считаются *родительскими (корнями)*, а $\mathbb{K}_{BC}^{agg,1\text{x}2}$ *дочерними ветвями parents* $K_B^{base}$ и $K_C^{base}$. У родителей может быть более одной дочерней ветви, хотя для ясности мы рассматриваем каждую копию родителя как отдельную. Это означает, что может быть несколько корней с одним и тем же ключом.

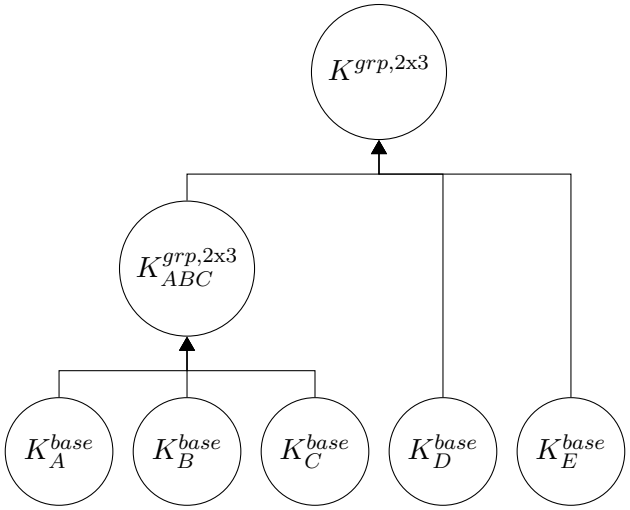
Например, в этих схемах «2 из 3» и «1 из 2», объединённых в «2 из 2», ключ Кэрол $K_C^{base}$ используется дважды и отображается дважды:




Для каждого подобъединения с мультиподписью определены отдельные наборы $\mathbb{S}$. В предыдущем примере есть три предварительно объединённых набора: $\mathbb{S}_{ABC}^{2\text{x}3} = \{K_{AB}^{sh,2\text{x}3}, K_{BC}^{sh,2\text{x}3}, K_{AC}^{sh,2\text{x}3}\}$, $\mathbb{S}_{CD}^{1\text{x}2} = \{K_{CD}^{sh,1\text{x}2}\}$, and $\mathbb{S}_{final}^{2\text{x}3} = \{\mathbb{K}_{ABC}^{agg,2\text{x}3}, \mathbb{K}_{CD}^{agg,1\text{x}2}\}$.

### 9.7.2 Вложение multisig-ключей

Предположим, у нас есть следующее семейство ключей



Если мы объединим ключи в $\mathbb{S}_{ABC}^{2\text{x}3}$, соответствующие первой схеме «2 из 3», мы столкнёмся с проблемой на следующем уровне. Давайте возьмем только один общий секрет между $K_{ABC}^{grp,2\text{x}3}$ и $K_D^{base}$, чтобы проиллюстрировать это:

$$k_{ABC,D} = \mathcal{H}_n(k_{ABC}^{grp,2\text{x}3} K_D^{base})$$

Теперь два участника из набора ABC легко могут внести компоненты ключей агрегирования, чтобы подобъединение могло вычислить

$$k_{ABC}^{grp,2\text{x}3} K_D^{base} = \sum k_{ABC}^{agg,2\text{x}3} K_D^{base}$$

Проблема состоит в том, что каждый участник из набора ABC может вычислить $k_{ABC,D}^{sh,2\text{x}3} = \mathcal{H}_n(k_{ABC}^{grp,2\text{x}3} K_D^{base})$! Если каждому из участников нижнего уровня будут известны все приватные ключи для создания мультиподписи более высокого уровня, то схемой multisig нижнего уровня также может быть «1 из N».

Эта проблема решается путём неполного объединения ключей до последнего дочернего ключа. Вместо этого мы просто применяем операцию `premerge` ко всем ключам, выводимым из нижнего уровня.

### Решение для вложения

Чтобы использовать $\mathbb{K}^{agg,\text{MxN}}$ в новой мультиподписи, мы передаем его так же, как и обычный ключ, но с одним изменением. Для операций, связанных с $\mathbb{K}^{agg,\text{MxN}}$, используется каждый из ключей-участников вместо объединённого ключа группы. Например, публичный «ключ» общего секрета между $\mathbb{K}_x^{agg,2\text{x}3}$ and $K_A^{base}$

$$\mathbb{K}_{x,A}^{sh,1\text{x}2} = \{[\mathcal{H}_n(k_A^{base}\mathbb{K}_x^{agg,2\text{x}3}[1]) * G], [\mathcal{H}_n(k_A^{base}\mathbb{K}_x^{agg,2\text{x}3}[2]) * G], ...\}$$

Таким образом, все участники $\mathbb{K}_x^{agg,2\text{x}3}$ знают только общие секреты, соответствующие их приватным ключам мультиподписи, построенной по схеме «2 из 3» нижнего уровня. Операция между набором ключей размером в два ${}^2\mathbb{K}_A$ и набором ключей размером в три ${}^3\mathbb{K}_B$ даёт набор ключей размером шесть ${}^6\mathbb{K}_{AB}$. Мы можем обобщить все ключи в семействе ключей как наборы ключей, где отдельные ключи обозначаются как ${}^1\mathbb{K}$. Элементы набора ключей упорядочиваются в соответствии с некоторым соглашением (то есть от наименьшего к наибольшему численно), а наборы, содержащие наборы ключей (например, наборы $\mathbb{S}$) упорядочиваются по первому элементу в каждом наборе ключей в соответствии с некоторым соглашением.

Мы допускаем распространение наборов ключей по структуре семейства, при этом каждая вложенная multisig-группа отправляет свой агрегированный набор до `слияния (merge)` в качестве «базового ключа» для следующего уровня,[33] пока не появится последний дочерний

[33] Следует отметить, что предварительное `слияние (merge)` необходимо выполнить для выходов *всех* вложенных мультиподписей, даже если мультиподписи, подстроенные по схеме «N' из N'», вложены в схему «N из N», потому что набор $\mathbb{S}$ изменится.

агрегированный набор, после чего наконец используется функция `слияния (merge)`.

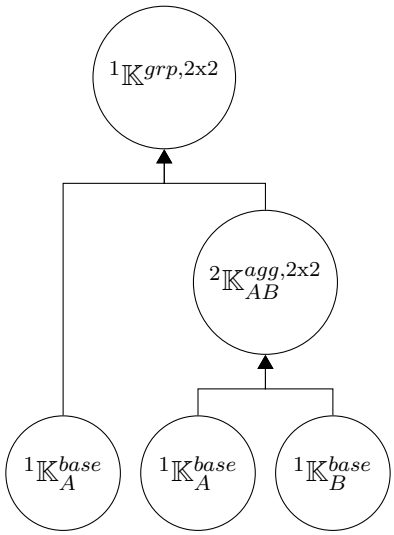
Пользователи должны хранить свои базовые приватные ключи, агрегированные приватные ключи для всех уровней структуры multisig-семейства и публичные ключи для всех уровней. Это облегчает создание новых структур, `merging (объединение)` вложенных мультиподписей и взаимодействие с другими подписантами для восстановления структуры в случае возникновения какой-либо проблемы.

### 9.7.3 Значение для Monero

Каждое подобъединение, вносящее свой вклад в создание окончательного ключа, должно вносить компоненты в транзакции Monero (например, начальные значения $\alpha G$) и поэтому каждое под-подобъединение также должно вносить свой вклад в своё дочернее подобъединение.

Это означает, что каждый родитель, даже если в структуре семейства имеется несколько копий одного и того же ключа, должен вносить один корневой компонент в свой дочерний элемент, а каждый дочерний элемент — один компонент уже в свой дочерний элемент, и так далее. Мы используем простые суммы на каждом уровне.

Например, возьмём это семейство




Скажем, им нужно вычислить некоторое групповое значение $x$ для подписи. Родители вносят свой вклад: $x_{A,1}$, $x_{A,2}$, $x_B$. Сумма будет следующей: $x = x_{A,1} + x_{A,2} + x_B$.

Схема вложенной подписи в настоящее время не реализована в Monero.

# глава 10

# Monero на рынке эскроу-услуг

В большинстве случаев покупки в интернет-магазинах проходят гладко. Покупатель отправляет деньги продавцу, а затем ожидаемый товар доставляется прямо к его порогу. Если у товара имеется какой-либо дефект или же, что происходит в некоторых случаях, покупатель меняет своё решение, товар может быть возвращён, а покупатель получит соответствующее возмещение.

Трудно доверять человеку или организации, с которыми вы не были знакомы ранее, и многие покупатели чувствуют себя в безопасности при совершении покупок только потому, что знают, что компания, выпустившая их кредитную карту, сможет отменить проведённый платеж по запросу [30].

Криптовалютные транзакции необратимы, и существует лишь ограниченное количество случаев, в которых покупатель или продавец могут воспользоваться средствами юридической защиты, если что-то пойдёт не так. Особенно это касается Monero, анализ блокчейна которой связан с определёнными трудностями [48]. Краеугольным камнем с точки зрения надёжности любого интернет-магазина, принимающего в качестве оплаты криптовалюты, является возможность прибегнуть к услугам эскроу-сервиса, использующего мультиподписи, которые строятся по схеме «2 из 3». Это позволяет третьим сторонам выступать в качестве посредника при разрешении споров. Доверяя этим третьим лицам, даже совершенно анонимные продавцы и их покупатели могут взаимодействовать без каких-либо формальностей.

Поскольку взаимодействие с использованием мультиподписей Monero подразумевает наличие определённых сложностей (см. подпункт 9.4.2), в этой главе будет говориться о максимально

эффективной среде для торговли посредством эскроу-сервисов.[1,2]

## 10.1 Важные особенности

Существует несколько основных требований и функций, упрощающих онлайн взаимодействие между покупателем и продавцом. Нами были использованы данные исследования OpenBazaar, проведённого Рене [117], поскольку они вполне разумны и расширяемы.

- *Офлайн продажи*: покупателю требуется доступ к онлайн-магазину продавца, чтобы разместить заказ, даже если продавец находится вне сети. Очевидно, что продавец должен войти в сеть, чтобы получить заказ и выполнить его.[3]

- *Платежи на основе заказа*: адреса продавцов для получения средств уникальны для каждого отдельного заказа. Это позволяет обеспечить соответствие заказов вносимым платежам.

- *Покупка при высоком уровне доверия*: покупатель, если он по-настоящему доверяет продавцу, может заплатить за товар ещё до того, как заказ будет выполнен или доставлен.
  - *Прямые онлайн-платежи*: убедившись в том, что продавец находится в сети и список его товаров доступен, покупатель отправляет продавцу деньги в одной транзакции на адрес, указанный продавцом, который затем сообщает покупателю, что заказ выполняется.
  - *Офлайн платежи*: если продавец находится вне сети, покупатель создаёт multisig-адрес по схеме «1 из 2» и перечисляет на него средства, достаточные, чтобы покрыть стоимость предполагаемой покупки. Когда продавец войдёт в сеть, он сможет снять деньги с multisig-адреса (отправив сдачу обратно покупателю) и выполнить заказ. Если продавец более уже не вернётся в сеть (или, например, по истечении некоторого разумного периода ожидания) или же если покупатель передумает, прежде чем вернётся продавец, покупатель сможет перевести средства обратно с адреса, созданного по схеме «1 из 2» обратно на свой личный кошелек.

- *Модерируемая покупка*: покупателем, продавцом и модератором создаётся multisig-адрес по схеме «2 из 3». Модератор при этом согласуется как покупателем, так и продавцом.

[1] Рене Бруннер “rbrunner7" является контрибьютором Monero, создавшим MMS [116, 115]. Именно им были проведены исследования в отношении возможности интеграции схем мультиподписи Monero в децентрализованную криптовалютную торговую площадку OpenBazaar `https://openbazaar.org/`. Изложенные здесь концепции непосредственно связаны с теми препятствиями, с которыми столкнулся Рене [117] (на «ранних этапах анализа»).

[2] Как нам кажется, в текущей реализации схемы мультиподписи Monero процесс создания транзакций уже аналогичен тому, который необходим для торговли с привлечением эскроу-сервисов, что представляется хорошей новостью с точки зрения потенциальной работы по её внедрению. Читателю следует обратить внимание на то, что схема мультиподписи Monero требует некоторого обновления в плане безопасности, прежде чем её можно будет широко использовать [102].

[3] Выполнение заказа подразумевает отправку товара, который будет доставлен покупателю.

Покупатель отправляет средства на этот адрес до выполнения своей части сделки продавцом, а затем, после доставки товара, обе стороны участвуют в процессе «разблокировки» средств. Если продавец и покупатель не смогут прийти к соглашению, они могут обратиться к модератору для разрешения спорной ситуации.

Мы не рассматриваем покупки при высоком уровне доверия, поскольку ввиду отсутствия необходимости в сложном взаимодействии процесс будет довольно тривиальным.[4]

### 10.1.1 Процесс покупки

Все покупки должны совершаться в соответствии с одним и тем же набором шагов при условии, что все стороны проявят при этом должную осмотрительность. Количество шагов зависит от того, с чем столкнётся модератор при вступлении в игру, например: «Вы запрашивали возврат средств, прежде чем привлечь меня?».

1. Покупатель получает доступ к онлайн-магазину продавца, выбирает товар, который собирается приобрести, выбирает опцию «Я хочу это приобрести», обеспечивает доступ к средствам, необходимым для покупки этого товара, а затем отправляет заказ продавцу.

2. Продавец получает заказ, проверяет, есть ли товар на складе и достаточно ли средств имеется у покупателя, и либо возвращает покупателю деньги, либо выполняет заказ, отправляя товар и уведомив об этом покупателя соответствующей квитанцией.

   - В случае с мультиподписью, созданной по схеме «2 из 3», покупатель может опционально подтвердить платёж при получении уведомления о выполнении заказа.

3. Покупатель либо получает товар должным образом, либо не получает товар вовремя, либо же получает товар с дефектом.

   - *Хороший товар*: покупатель либо оставляет отзыв для продавца, либо вообще не делает ничего.
     - *Покупатель оставляет отзыв*: покупатель может оставить как положительный, так и отрицательный отзыв.
       * *Положительный отзыв*: если используется схема мультиподписи «2 из 3» и товар ещё не был оплачен, то на этом этапе покупатель подтверждает перевод оплаты продавцу. В ином случае это просто положительный отзыв. [КОНЕЦ ПРОЦЕССА]
       * *Отрицательный отзыв*: если используется схема мультиподписи «2 из 3», то на этом этапе происходит переход к процессу «плохой товар». В ином случае это просто отрицательный отзыв.
     - *Покупатель ничего не делает*: либо продавцу уже заплатили, либо используется схема мультиподписи «2 из 3» и для получения средств ему требуется определённое взаимодействие с кем-то.

[4] Мультиподпись, построенная по схеме «1 из 2», может использовать преимущества некоторых концепций, полезных для создания мультиподписи по схеме «2 из 3», в частности, построение адреса в первую очередь.

    * *Продавцу заплатили*: [КОНЕЦ ПРОЦЕССА]
    * *Продавцу не заплатили*: продавец требует оплаты.

    (a) Продавец связывается с покупателем и запрашивает платёж (или отправляет напоминание).

    (b) Покупатель отвечает, либо не отвечает.

        · *Покупатель отвечает*: ответ покупателя может быть либо платежом, либо запросом возмещения.

        > *Покупатель производит платёж.* [КОНЕЦ ПРОЦЕССА]

        > *Покупатель запрашивает возврат средств*: переход к процессу «плохой товар».

        · *Покупатель не отвечает*: продавец обращается к модератору с целью «разблокировки» средств. [КОНЕЦ ПРОЦЕССА]

- *Отсутствие товара или плохой товар*: покупатель запрашивает возврат средств.

    (a) Покупатель связывается с продавцом и делает запрос о возврате средств, возможно, с объяснением причины.

    (b) Продавец выполняет запрос на возврат средств или оспаривает его (или же просто игнорирует его).

        – *Продавец выполняет запрос*: деньги возвращаются к покупателю. [КОНЕЦ ПРОЦЕССА]

        – *Продавец отказывается удовлетворить запрос*: используется либо схема multisig «2 из 3», либо иная схема.

            * *Используется иная схема от multisig «2 из 3»*: [КОНЕЦ ПРОЦЕССА]
            * *Используется схема multisig «2 из 3»*: либо покупатель отказывается от запроса на возврат средств (буквально или же он просто не может ответить своевременно), либо покупатель настаивает на возврате.

                · *Покупатель отказывается от запроса*: либо покупатель подтверждает платёж, либо нет.

                > *Покупатель подтверждает платёж*: [КОНЕЦ ПРОЦЕССА]

                > *Покупатель не подтверждает платёж*: продавец связывается с модератором, который подтверждает платёж. [КОНЕЦ ПРОЦЕССА]

                · *Покупатель настаивает на возмещении*: продавец или покупатель связывается с модератором, который обсуждает возникшую проблему вместе с участниками для вынесения решения. [КОНЕЦ ПРОЦЕССА]

## 10.2 Бесшовные мультиподписи Monero

Мы можем воспользоваться преимуществом, которое имеется при обычном заказе на приобретение товара, и «сжато» пройти почти все этапы взаимодействия с использованием мультиподписи Monero, построенной по схеме «2 из 3», так, что участники этого даже не заметят. Есть ещё один маленький шаг, который должен будет сделать продавец в конце. Ему будет

необходимо подписать и представить окончательную транзакцию для получения оплаты (по аналогии с «изъятием наличных из кассового аппарата»).[5]

### 10.2.1 Основные принципы взаимодействия в рамках схемы multisig

Все взаимодействия в рамках схемы multisig «2 из 3» предполагают наличие одинакового набора раундов обмена данными, включая создание адреса и построение транзакции. Для обеспечения соответствия нормальному рабочему процессу мы реорганизуем процесс построения транзакции, если сравнивать с тем, как это описано в главе, посвящённой мультиподписям (см. подпункты 9.4.2 и 9.6.2).[6]

1. *Создание адреса*

   (a) Прежде всего, все пользователи должны узнать базовые ключи других участников, которые будут использоваться ими для построения общих секретов. Затем они должны передать публичные ключи этих общих секретов (например, $K^{sh} = \mathcal{H}_n(k_A^{base} * k_B^{base} G)G)$ другим пользователям.

   (b) После того как все общие секретные публичные ключи будут известны, каждый пользователь может произвести предварительное слияние (функция `premerge`а затем окончательно объединить их (функция `merge`) в публичный ключ траты адреса. Агрегированные приватные ключи траты будут использоваться для подписания транзакций. Хеш общего секретного приватного ключа основных подписантов, покупателя и продавца, (например, $k^{sh} = \mathcal{H}_n(k_A^{base} * k_B^{base} G)$) будет использоваться в качестве ключа просмотра (например, $k^v = \mathcal{H}_n(k^{sh})$) будет использоваться в качестве ключа просмотра (например, 10.2.2).

2. *Построение транзакции*: предположим, что по адресу уже имеется хотя бы один выход, а образы ключей неизвестны. Есть два подписанта: инициатор и соподписант.

   (a) *Запуск создания транзакции*: инициатор решает создать транзакцию. Он генерирует начальные значения (например, $\alpha G$) для всех имеющихся у него выходов (он ещё не уверен, какие из них будут использованы) и создаёт по ним обязательства. Он также создаёт частичные образы ключей для принадлежащих ему выходов и подписывает их при помощи доказательства легитимности (см. подпункт 9.5). Затем он отправляет эту информацию вместе со своим личным адресом для получения средств (например, для получения сдачи, если это будет применимо, и т. д.) соподписанту.

[5] Расширение этой схемы за пределы «(N-1) из N», вероятно, невозможно без введения дополнительных шагов из-за наличия дополнительных раундов, необходимых для создания multisig-адреса с более низким пороговым значением.

[6] Данная процедура на самом деле очень похожа на то, как в настоящее время в Monero организуются транзакции с мультиподписями.

(b) *Создание частичной транзакции*: соподписант проверяет правильность всей информации, содержащейся в запущенной транзакции. Он определяет получателей выходов и соответствующие суммы (они могут быть частично основаны на рекомендациях инициатора), входы, которые будут использоваться вместе с соответствующими ложными участниками кольца, и комиссию за проведение транзакции. Он генерирует приватный ключ транзакции (или ключи, если используются подадреса), создаёт одноразовые адреса, обязательства по выходам, зашифрованные суммы, маски обязательств по псевдовыходам, и начальные значения обязательств по нулевой сумме. Чтобы доказать, что суммы находятся в пределах допустимого диапазона, он создаёт доказательства диапазона Bulletproofs для всех выходов. Он также генерирует начальные значения для своей подписи (но не даёт по ним обязательств), случайные скалярные величины для подписей MLSAG, частичные образы ключей для имеющихся у него выходов и доказательства легитимности этих частичных образов. Всё это отправляется инициатору транзакции.

(c) *Частичная подпись инициатора*: инициатор проверяет, является ли информация в частичной транзакции действительной и соответствует ли его ожиданиям (например, суммы и получатели те же, что и должны быть). Он заполняет подписи MLSAG и подписывает их своими приватными ключами, а затем отправляет частично подписанную транзакцию соподписанту вместе со своими раскрытыми начальными значениями.

(d) *Завершение создания транзакции*: соподписант завершает подписание транзакции и отправляет её в сеть.

### Подписание с одним обязательством

В отличие от того, что было рекомендовано нами в главе, посвящённой мультиподписям, для каждой частичной транзакции (инициатором транзакции) даётся только одно обязательство, и оно раскрывается после того, как соподписант уже точно отправит своё начальное значение. Цель обязательств по начальным значениям (например, $\alpha G$) состоит в том, чтобы помешать злоумышленнику использовать своё собственное начальное значение в попытке повлиять на создаваемый запрос, что потенциально позволило бы такому злоумышленнику обнаружить агрегированные ключи других соподписантов (см. подпункт 9.3). Если в момент, когда злоумышленник будет генерировать собственное начальное значение, хотя бы одно частичное начальное значение будет недоступно, то он не сможет (или по крайней мере сможет, но очень

малой вероятностью) контролировать запрос.[7,8,9]

Подобное упрощение позволяет убрать один раунд обмена данными, что имеет важные последствия с точки зрения взаимодействия покупателя и продавца.

### 10.2.2 Пользовательский опыт взаимодействия с эскроу-сервисами

В данном подпункте подробное рассматривается процесс взаимодействия покупателя, продавца и модератора при совершении онлайн-покупки за Monero с использованием мультиподписи, построенной по схеме «2 из 3».

1. *Покупатель совершает покупку*

   (a) *Новый сеанс со стороны покупателя*: покупатель входит на торговую онлайн-площадку, и его клиент генерирует новый подадрес, который будет использоваться, если покупатель начнёт оформление нового заказа на покупку.[10] На этой площадке он найдёт необходимых ему продавцов, и каждый продавец предложит на выбор свой товар и соответствующую цену. Каждый продукт имеет базовый ключ, используемый при совершении покупок по схеме multisig, и этот ключ невидим для

[7] По этой же причине инициатор раскрывает свои начальные значения только после определения всей информации транзакции. Поэтому ни один из подписантов не может изменить сообщение MLSAG и повлиять на запрос.

[8] Подписание с использованием единственного обязательства можно обобщить как подписание с (M-1)-обязательством, когда только автор частичной транзакции не производит обязательства и раскрытия, а другие соподписанты совершают раскрытие только после того, как информация транзакции будет определена полностью. Например, предположим, что есть адрес, построенный по схеме «3 из 3» соподписантами (A, B, C), которые попытаются подписаться с использованием одного обязательства. Подписанты B и C находятся в сговоре против A, при этом C является инициатором, а B — автором частичной транзакции. C инициирует транзакцию выдачей обязательства, а затем A выдаёт своё начальное значение (без обязательства). Когда B создаёт частичную транзакцию, он может вступить в сговор с C, чтобы контролировать запрос подписи в целях раскрытия приватного ключа A. Также следует обратить внимание на то, что подписание с (M-1)-обязательством — это оригинальная концепция, которая впервые предлагается именно в этой работе, и она не подкреплена какими-либо материалами передовых исследований или уже реализованным кодом. И эта концепция может оказаться совершенно ошибочной.

[9] Как вариант, можно рассматривать значение и цель «обязательства» (см. подпункт 5.1). Как вариант, можно рассматривать значение и цель «обязательства» (см. подпункт 5.1). Как только Элис создаст обязательство по значению A, она «застрянет» на этом и не сможет воспользоваться новой информацией, получаемой в результате события B (инициированного Бобом), которое происходит позже. Более того, если A не было раскрыто, оно не может повлиять на B. Элис и Боб могут быть уверены, что A и B не зависят друг от друга. Мы утверждаем, что подписание с использованием единственного обязательства, как было описано выше, соответствует этому стандарту и эквивалентно подписанию с полным набором обязательств. Если обязательство $c$ является односторонней функцией исходных значений $\alpha_A G$ и $\alpha_B G$ (например, $c = \mathcal{H}_n(\alpha_A G, \alpha_B G)$), то если обязательство по $\alpha_A G$ даётся изначально, $\alpha_B G$ раскрывается после появления $C(\alpha_A G)$, а $\alpha_A G$ раскрывается после появления $\alpha_B G$, то $\alpha_B G$ и $\alpha_A G$ независимы, а $c$ будет случайным как с точки зрения Элис, так и с точки зрения Боба (если они не будут взаимодействовать, или же с самой ничтожной вероятностью).

[10] Использование нового подадреса для каждого заказа или даже нового подадреса для каждого отдельного продавца или товара такого продавца затрудняет отслеживание продавцами поведения их заказчиков. Это также помогает убедиться в том, что каждый заказ товара уникален, в случае если кто-то покупает одно и то же дважды.

самого покупателя, но видим для его клиента (то есть программного обеспечения, которое он использует для совершения покупок). Наряду с этим базовым ключом также существует список предварительно выбранных модераторов, и каждый такой модератор имеет базовый ключ и предварительно вычисленный общий секретный публичный ключ, известный продавцу и модератору.[11]

(b) *Покупатель добавляет товар в корзину*: наш покупатель решает, что он что-то готов купить, выбирает вариант оплаты (например, прямой платеж с использованием мультиподписи по схеме «1 из 2» или мультиподписи по схеме «2 из 3»), и, если он выбирает схему multisig «2 из 3», ему предоставляется список доступных модераторов, из которого он может выбрать нужного. После того как покупатель добавит товар в свою корзину, клиент, невидимый для него (при условии, что была выбрана схема мультиподписи «2 из 3»), использует базовый ключ товара, базовый ключ модератора и общий секретный публичный ключ продавца и модератора, чтобы в сочетании с ключом траты собственного подадреса, созданного для этого сеанса (в качестве базового ключа), построить multisig-адрес покупателя-продавца-модератора по схеме «2 из 3».[12]

Ключ просмотра является хешем общего секретного приватного ключа покупателя и продавца (а не агрегированным приватным ключом, то есть ключом, созданным до предварительного `premerge (объединения)`), а ключ шифрования для обмена данными между покупателем и продавцом — это хеш ключа просмотра.[13]

(c) *Покупатель переходит к оформлению заказа*: покупатель просматривает корзину со всеми товарами и решает перейти к оформлению заказа. Именно на этом этапе он делает доступными свои средства ещё до завершения выполнения заказа. Его клиент строит транзакцию (но пока не подписывает её), которая будет либо прямым платежом продавцу, либо переводом средств на multisig-адрес (с небольшой доплатой за будущие комиссии). Если происходит перевод средств на multisig-адрес по схеме «2 из 3», клиент также инициирует две транзакции, отправляя деньги с этого адреса. Один может использоваться для оплаты продавцу, а другой - для возмещения покупателю. Частичные образы ключей основаны на транзакции перевода средств, которая еще не была подписана.

На самом деле, ему требуются только исходные значения с обязательствами для двух транзакций, а затем по отдельности одна копия частичных образов ключей (с

[11] Самым простым было бы продавцам также включить невидимые для покупателей обязательства по раскрытию значений для транзакций. Однако, чтобы обработать несколько заказов на покупку одного и того же товара, продавцу пришлось бы заранее предоставить множество обязательств для каждого потенциального покупателя. Можно только представить, насколько запутанным мог бы стать весь процесс. Это отчасти дополняет практичность, обеспечиваемую нашим упрощением подписания с использованием одного обязательства.

[12] То, как именно должна быть реализована торговая площадка, является предметом свободного решения, поскольку, например, выбор типа оплаты за товар может быть предоставлен пользователю при окончательном оформлении заказа, а не в интерфейсе «добавить в корзину».

[13] Тот же самый процесс будет использоваться в случае со схемой «1 из 2» без участия модератора.

доказательством легитимности) и одна копия подадреса для данного сеанса. Этот подадрес имеет двойное назначение: это адрес покупателя для возврата сфредств или получения выходов сдачи, а его ключ траты является базовым multisig-ключом покупателя.[14]

(d) *Покупатель решается на проведение оплаты*: после просмотра всех деталей заказа на товар покупатель разрешает оплатить его.[15] Его клиент завершает подписание транзакции перевода средств и отправляет её в сеть.[16] Он отправляет продавцу заказ на товар вместе с хешем транзакции перевода средств и общим секретным публичным ключом покупателя-модератора, а уже затем инициирует multisig-транзакции.[17]

2. *Продавец выполняет заказ на товар*

(a) *Продавец оценивает заказ на товар*: продавец изучает заказ на товар нашего покупателя, а затем утверждает его для отправки товара. Если ему заплатили напрямую, то рассматривать больше нечего, а если ему заплатили с использованием схемы multisig «1 из 2», то он может совершить транзакцию, заплатив себе с соответствующего адреса. В случае со схемой multisig «2 из 3» его клиент генерирует подадрес для получения суммы, указанной в заказе на товар, и производит две частичные транзакции из транзакций, инициированных покупателем. В транзакции платежа продавцу отправляется сумма, составляющая цену товара, а остальная часть отправляется покупателю в качестве сдачи, в то время как транзакция возмещения просто очищает multisig-адрес для покупателя.[18] Обратите внимание, что он реконструирует multisig-адрес на основе базовых ключей покупателя-поставщика-модератора в сочетании с общим секретным публичным ключом покупателя-модератора.

(b) *Продавец отправляет товар*: продавец отправляет товар, а также отправляет покупателю уведомление о выполнении заказа. Это уведомление включает в себя квитанцию о покупке, а также запрос о завершении платежа заказчиком (с этого момента всё относится к схеме multisig «2 из 3»). Подадрес заказа на товар продавца скрыт от пользователя, и он может использоваться в случае необходимости в разрешении спора, а также для построения обеих частичных транзакций.

[14] Важно создавать отдельные транзакции, поскольку исходные значения с обязательствами можно использовать только один раз.

[15] Если покупатель отменяет заказ на приобретение товара, его транзакция с переводом средств и частичные multisig-транзакции удаляются.

[16] Если в его корзине находились товары от нескольких продавцов, его клиент может создать несколько заказов на приобретение товара и обрабатывать их по отдельности. Все продавцы могут получать оплату посредством одной и той же транзакции с переводом средств.

[17] Клиент покупателя должен отслеживать детали платёжного поручения, такие как итоговая цена, чтобы впоследствии проверить содержание multisig-транзакций перед их подписанием.

[18] Частичные транзакции могут содержать много общих значений, поскольку они используют одни и те же входы, и только одна из них в конечном итоге должна быть подписана. Мы считаем, что из соображений сохранения модульности и надёжности конструкции лучше обрабатывать их по отдельности.

3. *Покупатель завершает проведение платежа или запрашивает возврат*: покупатель может сделать это, как только получит уведомление о выполнении заказа, или может дождаться доставки товара.

   (a) *Покупатель отправляет частично подписанную транзакцию*: покупатель решает, платить ли ему за свою покупку или запросить возврат средств. Его клиент создаёт частичную подпись для соответствующей частичной транзакции и отправляет её продавцу. Предполагается, что любой запрос возврата средств будет содержать объяснение, оправдывающее такой возврат.

   (b) *Продавец завершает транзакцию*: продавец получает частично подписанную транзакцию, завершает её подписание и отправляет в сеть. При необходимости он отправляет покупателю уведомление о возврате средств с доказательством.

4. *Модерируемый спор*: в любой момент после того, как покупатель отправит заказ на приобретение товара, и до того, как с multisig-адресов будут выведены средства, либо продавец, либо покупатель могут решить привлечь модератора к решению возможного спора. Сторона A — это тот, кто начал спор, а Сторона B — ответчик.[19]

   (a) *Сторона _A связывается с модератором*: сторона_A создаёт две транзакции для оплаты или возврата средств, на этот раз предназначенных для подписания Стороной_A-модератором, отправляет их модератору вместе с необходимой для создания multisig-адреса информацией (базовые ключи, общий секретный публичный ключ Стороны_A-Стороны_B и приватный ключ просмотра) и считывает баланс multisig-кошелька (частичные образы ключей и их доказательства).

   (b) *Модератор занимается разрешением спора*

      i. *Модератор признаёт наличие спорной ситуации*: модератор признаёт, что занимается рассмотрением спорной ситуации, и в то же время создаёт частичные транзакции из уже созданных транзакций и отправляет их Стороне_A. Он обязательно уведомляет Сторону_B о наличии спорной ситуации, а также создаёт ещё две транзакции со Стороной_B, чтобы предупредить невыполнение Стороной_A окончательного решения.

      ii. *Модератор выносит решение*: модератор просматривает доступные доказательства и может взаимодействовать со сторонами в случае необходимости сбора дополнительной информации. Он может попытаться и выступить посредником в разрешении спора в надежде, что обе стороны разрешат его без необходимости в вынесении решения.

      iii. *Спор подходит к концу*: либо покупатель и продавец разрешают спорную ситуацию самостоятельно, либо модератор выносит своё решение, которое сообщает обеим сторонам.

[19] Наша схема разрешения споров предполагает, что действующие лица будут действовать добросовестно. Люди, которые отказываются сотрудничать и, например, не инициируют, и не подписывают сделки, которые не выгодны им самим, без сомнения, сделают процесс намного более утомительным для всех участвующих сторон.

- Примечание: если сторона-ответчик, согласно вынесенному решению, должна получить средства, но по какой-либо причине не предоставила адрес, модератор может попытаться связаться и сотрудничать с ней, чтобы она получила эти средства. Поскольку сторона, инициировавшая спор, не участвует в данном процессе, такая связь может быть установлена (или продолжена) уже после разрешения спора.

(c) *Сторона_A или Сторона_B соглашается с решением*: если транзакции между Стороной_A и Стороной_B не были завершены, это означает, что спор был разрешён по решению модератора.

i. *Сторона_A соглашается с решением*

A. Сторона_A завершает свою частичную подпись в транзакции с решением и отправляет её модератору.

B. Модератор завершает подпись и отправляет транзакцию в сеть.

ii. *Сторона_B соглашается с решением*

A. Сторона_B создаёт частичную транзакцию для созданной транзакции с решением модератора и отправляет её модератору. Этот этап может быть выполнен до того, как решение станет окончательным, и в этом случае Сторона_B создаст частичные транзакции для обоих потенциальных возможных решений.

B. Модератор частично подписывает данную частичную транзакцию и отправляет её Стороне_B.

C. Сторона_B завершает подписание транзакции и отправляет её в сеть. Он отправляет хеш транзакции модератору.

(d) *Модератор закрывает спор*: модератор кратко излагает суть спора и своего решения и отправляет отчёт покупателю и продавцу.

Установлены четыре ключевых оптимизации технического решения.

### Предварительно выбранные модераторы

Заранее выбирая модераторов, продавцы могут создать общий секрет для каждого из них, для каждого из своих товаров, и опубликовать соответствующий публичный ключ с информацией о товаре.[20] Таким образом, покупатели могут создать полный объединённый multisig-адрес за один шаг, как только решат что-то купить, что полностью соответствует требованиям «офлайн-продажи». Предварительный выбор нескольких модераторов позволяет покупателям выбрать того, кому они больше доверяют.

Покупатели, если они не доверяют модераторам, выбранным продавцом, также могут взаимодействовать с выбранным уже ими онлайн-модератором, создавая multisig-адрес с базовым

[20] Важно отметить, что эти multisig-адреса по-прежнему устойчивы к тестированию агрегированных ключей, поскольку общие секреты покупателя неизвестны внешним наблюдателям.

ключом товара, предлагаемого продавцом. После получения заказа на приобретение товара продавец может принять нового модератора или отменить продажу.[21]

Мы ожидаем от покупателей и продавцов взаимного стремления к выбору хороших модераторов, чтобы со временем создать иерархию модераторов, организованную по качеству и справедливости предоставляемых ими услуг. Модераторы с более низким качеством предоставляемых услуг или модераторы с более низкой репутацией, скорее всего, будут зарабатывать меньше денег или обслуживать меньшее количество клиентов, или будут заниматься менее значимыми транзакциями.[22]

### Подадреса и идентификаторы товаров

Продавцы создают новый базовый ключ для каждой линейки товаров / каждого идентификатора, и эти ключи используются для создания multisig-адресов, построенных по схеме «2 из 3».[23] Когда продавцы выполняют заказ, они создают уникальный подадрес для получения средств, который также можно использовать для сопоставления заказов с полученными платежами.

Здесь эффективно выполняется требование к «платежам на основе заказа на приобретение товара», в частности потому, что средства, направляемые на разные подадреса, будут легко доступны из одного и того же кошелька (см. подпункт 4.3).

### Предварительные частичные транзакции

Транзакции с мультиподписями реализуются в несколько раундов, поэтому мы начинаем их проводить как можно скорее. Для удобства пользователя частичные транзакции, которые используются редко (например, транзакции возмещения), выполняются раньше, поэтому они сразу же доступны для подписания, если в них возникает необходимость.

[21] Из соображений удобства эскроу-сервис может «всегда быть онлайн», и вместо использования предварительно выбранных модераторов все multisig-адреса «2 из 3» могут создаваться таким сервисом при оформлении заказа на приобретение товара. Ещё вариант — использовать вложенную мультиподпись (см. подпункт 9.7), где предварительно выбранный модератор на самом деле является multisig-группой, созданной по схеме «1 из N». Таким образом, всякий раз при возникновении спора любой модератор из этой группы, который окажется доступным, сможет вмешаться. Для реализации такого варианта, вероятно, потребуются значительные усилия с точки зрения разработки.

[22] Нам пока неясно, какой или способ передачи средств будет наилучшим для модераторов или наиболее вероятно будет использоваться ими. Возможно, им будет выплачиваться фиксированная или процентная ставка за каждую модерируемую транзакцию или транзакцию, в которую они будут добавлены в качестве модератора (а затем, если комиссия не была предусмотрена в исходных частичных транзакциях, они будут просто отказываться от участия в разрешения спора) или же пользователи и/или поставщики, и/или торговые площадки будут заключать с ними договор.

[23] Этот базовый ключ также используется для покупки с использованием мультиподписей, построенных по схеме «1 из 2». Мы считаем важным не раскрывать приватный ключ траты в канале обмена данными, поэтому использование общего секрета покупателя и продавца в данном случае имеет вполне определённый смысл.

### Условный доступ модератора

Как для повышения эффективности, так и для повышения уровня анонимности модераторам требуется доступ к деталям сделки исключительно при урегулировании споров. Для достижения этой цели мы делаем приватный multisig-ключ просмотра хешем общего секретного приватного ключа покупателя-продавца $k^{v,grp}_{purchase-order} = \mathcal{H}_n(T_{mv}, k^{sh,2\text{x}3}_{AB})$, где $T_{mv}$ является разделителем домена ключа просмотра торговой площадки, A и B обозначают продавца и покупателя, соответственно, а $k^{sh,2\text{x}3}_{AB} = \mathcal{H}_n(k^{base}_A * k^{base}_B G)$. Мы предполагаем то, что он захочет «взглянуть» на скрипт обмена данными между покупателем и продавцом. Другими словами, ключом шифрования обмена данными является $k^{ce}_{purchase-order} = \mathcal{H}_n(T_{me}, k^{v,grp}_{purchase-order})$ (где $T_{me}$ является разделителем домена ключа шифрования торговой площадки).[24,25]

Модераторы получают доступ к обмену данными между покупателем и продавцом и возможность подтверждать платежи только тогда, когда одна из первоначальных сторон выдаёт им ключ просмотра.[26]

Кроме того, продавцы могут проверить, не является ли хост торговой площадки (который также может быть доступен только для модераторов в зависимости от того, как реализуется данная концепция) MITM («атакой посредника») их разговоров с заказчиками (то есть не притворяется ли он покупателем или продавцом). Это делается путём проверки соответствия базовых ключей, публикуемых ими для каждого товара, тому, что отображается. Поскольку базовый ключ покупателя, который используется для создания multisig-адреса, также является частью ключа шифрования, вредоносному хосту будет трудно организовать MITM-атаку.

[24] Разделение ключа просмотра и ключа шифрования позволяет давать права только на просмотр журнала обмена данными без права просмотра истории транзакций конкретного multisig-адреса.

[25] Этот метод также используется в случае с multisig-адресами. созданными по схеме «1 из 2».

[26] Модераторам важно убедиться в том, что журнал обмена данными, который они получают, не был подделан. Одним из способов является включение каждым соподписантом подписанного хеша текущего журнала обмена данными при отправлении нового сообщения. В этом случае модераторы смогут просмотреть ряд хешированных журналов и выявить возможные расхождения. Это также помогло бы соподписантам выявить сообщения, которые не были переданы, и в качестве альтернативы создать доказательства того, что конкретные подписанты действительно получили определённые сообщения.

# глава 11

# Объединённые транзакции Monero (TxTangle)

Существует целый ряд неизбежных способов эвристического анализа графов транзакций, используемых в зависимости от объектов, по отношению к которым они применяются, и сценариев. В частности, поведение майнеров, пулов (см. подраздел 5.1 в работе [93]), торговых площадок, использующих эскроу-счета, а также бирж соответствует определённым шаблонам, которые уязвимы для анализа даже в случае с основанным на применении кольцевых подписей протоколом Monero.

В данном разделе мы описываем TxTangle, аналогичный CoinJoin [2], который используется Bitcoin, метод, позволяющий избежать такого эвристического анализа.[1] По сути, происходит слияние нескольких транзакций в одну, что смешивает поведенческие шаблоны каждого из участников.

Чтобы обеспечить эффективность такой обфускации, необходимо сделать так, чтобы внешним наблюдателям было неоправданно сложно использовать данные объединённых транзакций с целью определения групп входов и выходов для их дальнейшей привязки к отдельным участникам, а также нужно, чтобы эти наблюдатели в принципе не могли узнать, сколько пользователей участвует в создании транзакции.[2] Более того, даже самим участникам не

[1] В этой главе предлагается протокол создания объединённых транзакций. На момент написания документа такого протокола ещё не существовало. Предыдущее предложение под названием MoJoin было создано исследователем из Исследовательской лаборатории Monero под псевдонимом Sarang Noether (Саранг Ноезер) и требовало участия доверенного лица. Это противоречит базовому принципу проекта Monero, связанному с соблюдением анонимности и обеспечением взаимозаменяемости, и поэтому работа над MoJoin не была продолжена.

[2] Поскольку любой может увидеть суммы транзакций Bitcoin, как правило, существует возможность объединения входов и выходов CoinJoin, исходя из соответствующих сумм. [34]

должно быть известно их точное количество, и они не должны иметь возможности объединения входов и выходов других участников в группы, если только не контролируются все группы входов и выходов, кроме группы одного участника.[3] Наконец должна иметься возможность строить объединённые транзакции без участия какой-либо централизованной контролирующей процесс стороны [58]. К счастью, Monero обеспечивает соответствие всем этим требованиям.

## 11.1 Построение объединённых транзакций

В обычной транзакции входы и выходы связаны между собой при помощи доказательства того, что суммы сбалансированы. Как было указано в подпункте 6.2.1, сумма обязательств по псевдовыходам равна сумме обязательств по выходам (плюс обязательство по комиссии).

$$\sum_j C'^a_j - (\sum_t C^b_t + fH) = 0$$

Простая объединённая транзакция может включать в себя всё содержимое множества транзакций и позволяет объединить их в одну. Сообщения MLSAG могли бы использоваться для подписания всех данных подтранзакций, и сбалансированность сумм очевидно бы работала ($0 + 0 + 0 = 0$). Простая объединённая транзакция может включать в себя всё содержимое множества транзакций и позволяет объединить их в одну. Сообщения MLSAG могли бы использоваться для подписания всех данных подтранзакций, и сбалансированность сумм очевидно бы работала ($0 + 0 + 0 = 0$).[4] Однако, тогда группу входов и выходов можно было бы выявить путём определения соответствия поднаборов входов/выходов суммам.[5]

Эта проблема легко решается путём вычисления общих секретов каждой пары участников и последующего добавления этих смещений масок обязательств по псевдовыходам (см. подпункт 5.4). В каждой паре один из участников добавляет общий секрет к одному из *своих* обязательств по псевдовыходу. После того как всё будет просуммировано, секреты будут отменены, и поскольку у каждой пары участников имеется общий секрет, сумма будет сбалансирована только после того, как все обязательства будут объединены.[6]

Общие секреты позволяют скрыть группу входов/выходов в прямом смысле, но участникам необходимо как-то узнать все входы и выходы, и проще всего это сделать, если они сами сообщат свои группы входов/выходов. Очевидно, это нарушит изначальную задумку и будет подразумевать, что участникам будет известно их общее количество.

[3] В случае с данным методом существует риск проведения атаки путём умышленного «засорения» объединённых транзакций, что впервые было обнаружено в CoinJoin. [91]

[4] Поскольку доказательства в стиле Bulletproofs, по сути, объединяются в одно (см. подпункт 5.5), даже в самом простом случае от участников потребуется некоторое сотрудничество.

[5] Участники могут хитрым образом разделить комиссию, чтобы запутать наблюдателей. Но это не сработает в случае атаки методом грубого перебора, так как размер комиссий недостаточно велик (примерно 32 бита или менее).

[6] Вместо обязательств по выходам мы смещаем обязательства по псевдовыходам, так как маски обязательств по выходам строятся на основе адреса получателя (см. подпункт 5.3).

### 11.1.1 $n$-направленный канал передачи данных

Максимальное количество участников транзакции TxTangle определяется либо количеством выходов, либо количеством входов (в зависимости от того, какое значение будет ниже). В рамках нашей модели каждый реальный участник притворяется, что он является другим человеком в случае с каждым отправляемым им выходом. В первую очередь это делается с целью создания группового канала передачи данных с другими потенциальными участниками, но общее количество участников при этом не раскрывается.

Представьте, что $n$ ($2 \leq n \leq 16$, хотя рекомендуется наличие по крайней мере 3)[7] предположительно не связанных друг с другом людей собирается со случайными интервалами в чатруме, который откроется во время $t_0$, близкое к $t_1$ (одновременно в комнате может присутствовать только 16 человек, и приоритет в ней отдаётся на основе комиссий, базовых комиссий на байт [для простоты согласования текущей средней величины и вознаграждения за блок], а также диапазона допустимых типов транзакций, поскольку, например, транзакции, которые были проведены в начале существования Monero, нельзя напрямую потратить в транзакции RingCT [132]). С наступлением $t_1$ все фиктивные участники заявляют о своём желании продолжить процедуру, публикуя публичные ключи, и комната превращается в $n$-направленный канал передачи данных, когда создаётся общий секрет для всех её фиктивных участников.[8] Этот общий секрет используется для шифрования содержимого сообщений, в то время как фиктивные участники подписывают связанные со входами сообщения, используя подписи SAG (см. подпункт 3.3). Поэтому остаётся совершенно неясным, кто отправил отдельно взятое сообщение. Сообщения, связанные с выходами, подписываются при помощи bLSAG (см. подпункт 3.4) в соответствии с набором публичных ключей фиктивных участников. Таким образом, фактические выходы не получится связать с ними.[9]

### 11.1.2 Количество раундов обмена сообщениями при построении объединённой транзакции

После того как канал будет настроен, можно строить транзакции TxTangle. Это делается за пять раундов обмена данными, где очередной раунд может начаться только после завершения предыдущего, и для выполнения каждого раунда даётся определенный временной интервал, в течение которого сообщения публикуются случайным образом. Эти интервалы необходимы во избежание создания кластеров сообщений, которые могут раскрыть группы входов/выходов.

1. Каждый фиктивный участник анонимно генерирует случайную скалярную величину для каждого предполагаемого выхода и подписывает их при помощи bLSAG. Отсортированный список этих скалярных величин используется для определения индексов выходов

[7] В настоящее время транзакция может содержать 16 выходов максимум.

[8] Метод мультиподписи, о котором говорится в подпункте 9.6.3, является однонаправленным, и схема «M-из-N» будет безоговорочно расширяться до «1-из-N».

[9] В каждом отдельном наборе bLSAG транзакций TxTangle должны использоваться одни и те же образы ключей, чтобы всё относящееся к конкретному выходу можно было связать.

(см. подпункт 4.2.1; наименьшая скалярная величина получает индекс $t = 0$).[10] Они публикуют эти bLSAG, а также SAG, которыми подписываются номера версии транзакции для заданных входов. После прохождения этого раунда участники могут вычислить вес транзакции, исходя из количества входов и выходов, а также точно определить размер необходимой комиссии.[11,12,13]

2. Каждый фиктивный участник использует ряд публичных ключей, чтобы создать общий секрет с другим участником с целью смещения их обязательств по псевдовыходам, а также решает, кто будет добавлять, а кто вычитать в зависимости от того, какой из публичных ключей в каждой паре будет меньше.[14] Каждый из фиктивных участников должен заплатить $1/n$ от вычисленной комиссии (при помощи целочисленного деления). Фиктивный участник с самым низким значением индекса выхода несёт ответственность за выплату остатка после деления (это будет действительно ничтожная сумма, но её необходимо учитывать во избежание «маркировки» транзакций TxTangle). Участники анонимно генерируют публичные ключи транзакций для каждого из своих выходов (пока что не для отправки другим участникам) и создают обязательства по своим выходам, зашифрованные суммы, частичные доказательства Части A, которые будут использоваться для создания совокупного доказательства диапазона Bulletproof. Всё это подписывается при помощи bLSAG (одно обязательство, одна зашифрованная сумма и одно частичное доказательство на сообщение bLSAG, а образ ключа связывает этот оригинальный список случайных скалярных величин, которые использовались для обозначения индексов выходов). Обязательства по псевдовыходам генерируются обычным образом (см. подпункт 5.4), а затем смещаются при помощи общих секретов и подписываются с использованием SAG. После того как bLSAG и SAG будут опубликованы, а общая комиссия в одинаковой степени верно будет рассчитана всеми участниками, можно верифицировать общую сбалансированность сумм.[15]

[10] Выбор индексов выходов должен соответствовать другим вариантам реализации конструкций транзакций. Это позволит избежать «маркировки» транзакций другим программным обеспечением. Мы используем этот случайный подход, чтобы обеспечить соответствие основному варианту реализации, в котором индексы выходов также выбираются случайным образом.

[11] Если в результате сравнения количества входов и выходов с чьи-то собственным набором входов/выходов выяснится, что в создании транзакции TxTangle участвуют всего два человека, то создание такой транзакции лучше прекратить. Рекомендуется, чтобы у каждого участника было по крайней мере два входа и два выхода на тот случай, если какой-либо злоумышленник решит не прекращать создание транзакции TxTangle, даже зная о наличии всего двух участников. Данная рекомендация подлежит дальнейшему обсуждению, поскольку использование большего количества входов и выходов не является нейтральным с точки зрения проведения эвристического анализа.

[12] В дополнительном поле транзакций TxTangle не должно присутствовать какой-либо внешней информации (например, незашифрованных идентификаторов, если только это не транзакция TxTangle с 2 выходами, где идентификатор платежа должен быть хотя бы фиктивно зашифрован).

[13] При оценке размера комиссии должен использоваться стандартизированный подход, чтобы у каждого из участников получался один и тот же результат. В противном случае могут образоваться кластеры выходов, основанные на методе вычисления. Тот же стандарт вычисления размера комиссии должен использоваться и в случае с транзакциями, не являющимися TxTangle, чтобы транзакции TxTangle выглядели так же, как и обычные транзакции.

[14] Так как точки сжаты (см. подпункт 2.4.2), ключи интерпретируются как целые 32-байтовые числа. По определению владелец самого малого ключа складывает, а владелец самого большого — вычитает.

[15] Мы не публикуем отдельные суммы комиссий, выплачиваемых в том случае, если участник вычислил её

3. Если суммы сбалансированы надлежащим образом, можно начинать дополнительный раунд построения совокупного доказательства Bulletproof, которое докажет, что все суммы в выходах находятся в пределах допустимого диапазона. Каждый фиктивный участник использует частичные доказательства Части A предыдущего раунда и соответствующие обязательства по выходам и анонимно вычисляет совокупный запрос A. Они используются для построения собственного частичного доказательства Части B, которое отправляется по каналу вместе с bLSAG.

4. Участники начинают заполнять сообщение, которое будет подписано при помощи MLSAG (см. сноску в подпункте 6.2.2). Два вида сообщений публикуются в случайном порядке через интервал связи. Каждое смещение участника кольца с выходом и каждый образ ключа подписывается при помощи SAG и связывается с правильным обязательством по псевдовыходу. Одноразовый адрес каждого выхода, публичный ключ транзакции и частичное доказательство Части C (вычисляется на основе частичных доказательств Части B и совокупного запроса B) подписываются при помощи bLSAG (сюда также может входить случайный компонент публичного ключа базовой транзакции, который, как мы увидим, можно задействовать во избежание подделки путём проведения атаки Януса).

5. Участники используют все частичные доказательства, чтобы создать совокупное доказательство Bulletproof, и анонимно используют метод логарифмического скалярного произведения, чтобы сжать его до окончательного доказательства, которое будет включено в данные транзакции. Как только вся информация, подлежащая подписанию MLSAG, будет собрана, каждый из участников должен создать MLSAG для своих входов и случайным образом отправить их (применив SAG к каждому) по каналу с интервалом связи. Любой из участников может передать свою транзакцию, как только будут собраны все её части.

### Публичные ключи транзакций и способ избежать атаки Януса

Если каждому из участников транзакции TxTangle известен приватный ключ транзакции $r$ (см. подпункт 4.2), то любой из них может сравнить одноразовые адреса выходов со списком известных адресов. Поэтому существует необходимость в построении транзакций TxTangle, как если бы был получатель с подадресом (см. подпункт 4.3), включая использование различных публичных ключей транзакции для каждого выхода.

В целях соответствия возможным вариантам предотвращения возможности проведения атаки Януса, связанной с подадресами, когда в дополнительное поле включается дополнительный «базовый» публичный ключ транзакции [101], TxTangle также должна содержать фальшивый

неверно, что может раскрыть кластер выходов из-за накопления ряда нестандартных сумм комиссии. Если суммы не сбалансированы надлежащим образом, создание транзакции TxTangle можно прервать.

«базовый» ключ, состоящий из суммы случайных ключей, сгенерированных каждым из фиктивных участников.[16]

У множества участников TxTangle, отправляющих деньги на подадрес, вероятнее всего, будет, по крайней мере, по два выхода, один из которых будет использоваться, чтобы вернуть сдачу участнику. Это означает, что любой из участников TxTangle может обеспечить защиту от атаки Януса, также сделав публичный ключ транзакции сдачи «базовым» ключом для подадреса получателя.[17] Получатель, использующий подадрес, может понять, что транзакция является TxTangle и что «базовый» ключ, вероятно, соответствует выходу сдачи отправителя.[18]

### 11.1.3 Слабые стороны

У злоумышленников есть два основных способа убить смысл TxTangle, то есть обойти механизм сокрытия групп входов/выходов от потенциального анализа. Они могут «засорить» транзакции так, что ряд честных участников станет предельно мал (или их вообще не будет) [91]. Они также могут воспрепятствовать попытке создания TxTangle и использовать последующие попытки тех же участников, чтобы оценить группы входов/выходов.

В первом случае этого будет сложно избежать, особенно при децентрализованном сценарии, где ни один из участников не будет обладать достаточной репутацией. Одним из способов применения транзакций TxTangle является их использование в совместных пулах, которые скрывают, к какому именно пулу из целого набора принадлежит майнер. Таким пулам будут известны группы входов/выходов, но так как их целью является помощь подключившимся к ним майнерам, это будет подталкивать их к тому, чтобы хранить эту информацию в секрете. Более того, такие транзакции TxTangle исключат из процесса злоумышленников, если допустить, что поведение пулов будет честным.

В последнем случае защиту можно обеспечить, только попытавшись провести TxTangle несколько раз перед тем, как прервать её, и при этом в случае с каждой транзакцией всегда необходимо генерировать большее количество случайных элементов. Среди этих элементов

[16] Отмена ключа (см. подпункт 9.2.2) не должна быть проблемой, поскольку это всего лишь фальшивый ключ, и в идеале его следует случайным образом проиндексировать в списке публичных ключей транзакции.

[17] При отправке средств на ваш собственный подадрес нет никакой необходимости в предотвращении атаки Януса. Кошельки с включенной защитой от такого вида атаки должны распознавать, что сумма, которая тратится в транзакции TxTangle, равна сумме, полученной на ваш подадрес, и поэтому они ошибочно не уведомляют пользователя о возможной проблеме.

[18] Предполагается, что количество публичных ключей транзакции совпадает в соотношении 1:1 с выходами, как, очевидно, и делается сегодня. Если бы по стандарту публичные ключи транзакции располагались в дополнительном поле в случайном или отсортированном порядке, то транзакции TxTangle и транзакции другого вида были бы в значительной степени неотличимы для получателей, использующих подадреса. Существуют особые случаи, когда участники TxTangle не могут включить «базовый» ключ (например, когда все их выходы относятся к подадресам), или когда транзакция явно не относится к виду TxTangle, поскольку получатель, использующий подадрес, получает большую часть или выходов, или всех их. Следует отметить, что, поскольку транзакции TxTangle обычно имеют намного больше выходов, чем типичная транзакция, для того чтобы отличить TxTangles от обычных транзакций с подадресами, можно прибегнуть к эвристическому анализу.

публичные ключи транзакции, маски обязательств по псевдовыходам, скалярные величины доказательств диапазона и скалярные величины MLSAG. В частности, набор ложных выходов в кольцах должен оставаться тем же, что позволит избежать перекрёстного сравнения, позволяющего выявить действительный вход. Если это возможно, то при разных попытках создания TxTangle следует использовать и разные действительные входы. Поскольку такая уязвимость неизбежна, она делает концепцию, изложенную в следующем подпункте, более важной.

## 11.2 Организованная транзакция TxTangle

В случае с действительно децентрализованными транзакциями TxTangle остаётся несколько открытых вопросов. Как запускаются и реализуются рассчитанные по времени раунды? Как вообще создаются чатрумы, чтобы участники могли найти друг друга? Самый простой способ — организовать хост TxTangle, который будет генерирует эти чатрумы и управлять ими.

Такой хост, казалось бы, сводит на нет цель, заключающуюся в сокрытии участия, поскольку каждый человек должен будет подключиться к нему и отправлять сообщения, которые можно затем использовать для корреляции групп входов/выходов (особенно в том случае, если хост участвует в создании транзакции и знает содержимое сообщения). Мы могли бы использовать такую сеть, как I2P[19], чтобы каждое сообщение, полученное хостом, выглядело так, как если бы оно было отправлено уникальным пользователем.

### 11.2.1 Базовые принципы обмена данными с хостом через I2P и другие функции

С помощью I2P пользователи создают так называемые «туннели», по которым передаются зашифрованные сообщения. Перед тем как достичь адресата, сообщения проходят через клиентов других пользователей. Из чего можно понять, что по этим туннелям может быть передано множество сообщений, прежде чем они будут уничтожены и воссозданы (например, для таких туннелей может быть установлен 10-минутный таймер). В нашей ситуации важно тщательно контролировать, когда создаются новые туннели и какие сообщения могут поступать из одного и того же туннеля.[20]

1. *Применительно к TxTangles*: в нашем первоначальном $n$-направленном варианте (см. подпункт 11.1.1) фиктивные участники постепенно добавляются в доступные комнаты

[19] The Invisible Internet Project - Проект «Невидимый интернет» (https://geti2p.net/en/).

[20] В I2P есть «исходящие» и «входящие» туннели (см. https://geti2p.net/en/docs/how/tunnel-routing). Всё, что принимается по входящим туннелям, выглядит так, как будто получено из одного и того же источника, даже если на самом деле их несколько. Поэтому на первый взгляд может показаться, что пользователям TxTangle не нужно создавать разные туннели для всех сценариев использования. Однако если хост TxTangle делает себя точкой входа для собственного входящего туннеля, то он получает прямой доступ к исходящим туннелям участников TxTangle.

TxTangle до того, как они будут закрыты. Однако если достаточно большое количество пользователей одновременно попытается создать TxTangle, высока вероятность сбоя, поскольку пользователи попытаются случайным образом поместить все свои выходы в одну и ту же «комнату» TxTangle, но тогда комнаты будут заполняться слишком быстро, и пользователям придётся отступить. Это вызвало бы самую настоящую путаницу.

Мы можем провести эффективную оптимизацию, сообщив хосту, сколько выходов у нас имеется (например, предоставив ему список наших публичных ключей фиктивного участника), и позволив ему собрать всех участников TxTangle. Поскольку мы по-прежнему следуем протоколу обмена сообщениями bLSAG и SAG, хост не сможет идентифицировать группы выходов в окончательной транзакции. Всё, что ему будет известно — это количество участников и количество выходов, имеющихся у каждого из них. Более того, при таком сценарии наблюдатели не смогут отслеживать открытые комнаты TxTangle с целью получения информации об участниках, что является важным улучшением с точки зрения анонимности. Обратите внимание, что способность хоста «засорять» TxTangles не отличается существенно от решения без привлечения хоста, поэтому это изменение будет нейтральным для данного вектора атаки.

2. *Метод передачи данных*: поскольку хост уже действует как локус передачи сообщений, он без труда может управлять передачей данных TxTangle. Во время каждого раунда хост собирает сообщения от фиктивных участников (всё ещё случайным образом в течение определенного временного интервала связи), а в конце раунда имеется короткая фаза распределения данных, когда он отправляет все собранные данные каждому участнику с определённым периодом буферизации. Это делается перед следующим раундом, чтобы убедиться в том, что сообщения получены и у участников есть время для их обработки.

3. *Туннели и группы входов/выходов*: после того как создание транзакции TxTangle будет инициировано, пользователям будет необходимо отделить свои фиктивные личности от фактических создаваемых выходов. Это означает необходимость в создании новых туннелей для передачи сообщений, подписанных bLSAG, и по каждому такому туннелю могут передаваться только сообщения, относящиеся к определённому выходу (можно передавать несколько таких сообщений через один и тот же туннель, поскольку очевидно, что информация об одном и том же выходе поступает из одного и того же источника). Также следует создать новые туннели для подписанных SAG сообщений, относящихся к определённым входам.

4. *Угроза проведения атаки посредника (MITM) хостом*: хост может обмануть участника, притворившись другими участниками, поскольку он контролирует рассылку списка фиктивных участников для создания bLSAG и SAG. Другими словами, список, который он отправляет пользователю A, может содержать фиктивных участников пользователя A, а все остальные будут его собственными. Сообщения, получаемые пользователем B, отклоняются при помощи списка A перед повторной передачей пользователю A. Поскольку все сообщения, подписанные с использованием списка A, принадлежат A,

хост будет иметь непосредственное представление о группах входов/выходов A!

Мы можем запретить хосту действовать в качестве MITM при взаимодействии с честными участниками, изменив способ создания публичных ключей транзакции. Участники отправляют друг другу свои предполагаемые публичные ключи транзакции обычным образом (с помощью bLSAG), а затем, как и в случае надёжной агрегации ключей, описанной в подпункте 9.2.3, к фактическим ключам, которые включаются в данные транзакции (и используются для создания масок обязательств по выходам и т. д.) в начале добавляется хеш списка фиктивных участников. Другими словами, $\mathcal{H}_n(T_{agg}, \mathbb{S}_{mock}, r_t G) * r_t G$ является публичным ключом $t$-й транзакции. Включение списка фиктивных участников в саму транзакцию затрудняет завершение TxTangles без прямого взаимодействия между всеми фактическими участниками.[21]

### 11.2.2 Использование хоста в качестве сервиса

Для обеспечения надёжности и постоянного улучшения важно, чтобы сервис TxTangle использовался с целью извлечения прибыли.[22] Вместо того чтобы ставить под угрозу идентификационные данные пользователей, получаемые с помощью модели на основе учётных записей, хост может участвовать в каждой транзакции TxTangle, давая единственный выход, и требовать от участников, чтобы те его финансировали. При получении доступа к сайту eepsite/сервису хоста для создания TxTangles пользователи также получают и уведомление о текущей стоимости хостинга, которую им будет необходимо уплатить за каждый выход.

Участники будут нести ответственность за оплату части комиссии $u$ сбора за организацию хоста. На этот раз самый малый ключ фиктивного участника (за исключением ключа хоста) будет должен оплатить оставшуюся часть комиссии и сбора за организацию хоста.[23] Поскольку у хоста нет входов, у него нет и обязательств по псевдовыходам, чтобы отменить маску обязательства по своему выходу. Вместо этого он, как обычно, вместе с другими фиктивными участниками создает общие секреты, а затем разделяет свою настоящую маску обязательства на части произвольного размера для каждого фиктивного участника и делит их на общие секреты. Он публикует список этих скалярных величин (соотнося их с каждым фиктивным участником на основе их публичных ключей), подписываясь своим ключом фиктивного участника, чтобы остальные пользователи узнали его от хоста. Появление этого списка сигнализирует о начале раунда 1, описанного в подпункте 11.1.2 (например, об окончании раунда настройки «0»). Фиктивные участники умножают свою скалярную величину, полученную от хоста, на соответствующий общий секрет и добавляют её к своей маске обязательства по

[21] Если реализуется защита от атаки Януса, вместо этого должна выполняться данная защита от MITM, что делается с помощью фальшивого базового ключа, используемого для предотвращения атаки Януса. Каждый фиктивный участник выдаёт случайный ключ $r_{mock}G$, и тогда фактическим базовым ключом будет $\sum_{mock} \mathcal{H}_n(T_{agg}, \mathbb{S}_{mock}, r_{mock}G) * r_{mock}G$.

[22] Несмотря на то, что развёрнутый сервис TxTangle может использоваться для извлечения прибыли, сам код может быть открытым. Этот аспект важен с точки зрения аудита программного обеспечения кошелька, которое взаимодействует с сервисом TxTangle.

[23] В данном случае мы должны использовать ключи фиктивного участника, поскольку хост не платит комиссию, а индекс его выхода неизвестен.

псевдовыходу. Таким образом, даже выход хоста не может быть идентифицирован каким-либо из участников окончательной транзакции, если только все они не объединятся против него.

Чтобы упростить расчёт комиссии, хост может распределить общую комиссию, которая будет использоваться в рамках транзакции, в конце первого раунда, так как он узнает вес транзакции раньше. Участники могут убедиться, что сумма соответствует ожидаемой, и оплатить свою долю.

Если участники объединятся, чтобы обмануть хост, и не внесут плату за хостинг, то хост может завершить проведение TxTangle в 3-м раунде. Он также может завершить процесс, если в канале появляются сообщения, которых там не должно быть или же которые окажутся недействительными.

В конце 5-го раунда хост завершает транзакцию и отправляет её в сеть для верификации, что является частью предоставляемых им услуг. Он включает хеш транзакции в окончательное сообщение о распространении.

## 11.3 Пользование услугами проверенного посредника

У децентрализованной модели TxTangle есть некоторые недостатки. Она требует, чтобы все участники активно общались в рамках строгого графика (и для начала находили друг друга), что сложно реализовать.

Привлечение центрального посредника, который бы отвечал за сбор информации о транзакциях каждого участника и производил бы обфускацию групп входов/выходов, может упростить процедуру. Ценой в данном случае является более высокий уровень доверия, поскольку посредник должен (как минимум) знать эти группы.[24]

### 11.3.1 Процедура с привлечением посредника

Посредник должен активно предлагать свои услуги по управлению созданием TxTangles и собирать заявки от потенциальных участников (состоящие из количества предполагаемых входов [с их типами] и выходов). При желании он может участвовать в процессе, используя свой собственный набор входов/выходов.

После того как группа, состоящая почти из 16 выходов, будет собрана (должно быть два или большее количество участников, и ни один участник не может обладать всеми наборами, кроме одного выхода или входа), посредник запускает первый из пяти раундов. При проведении каждого раунда он собирает информацию, получаемую от каждого участника, принимает некоторые решения и рассылает сообщения, которые означают начало нового раунда.

[24] На написание данного подпункта нас вдохновила схема протокола MoJoin.

1. Сначала для каждой пары участников посредник генерирует случайную скалярную величину и решает, кому в каждой паре должна принадлежать положительная или отрицательная версия. Чтобы оценить размер общей требуемой комиссии, он использует количество и тип входов и выходов. Он суммирует скалярные величины каждого пользователя и анонимно отправляет каждому их сумму вместе с указанием той части комиссии, которую они должны заплатить, и (случайно выбранными) индексами их выходов. Эти сообщения представляют собой сигнал участникам о начале проведения TxTangle.

2. Каждый участник строит свою подтранзакцию, как обычно, генерируя отдельные публичные ключи транзакции для своих выходов (с уменьшением риска проведения атаки Януса по мере необходимости), вычисляя одноразовые адреса выходов и кодируя суммы в выходах, создавая обязательства по псевдовыходам, которые сбалансированы с обязательствами по выходам и частью комиссии. Так составляется список смещений участников кольца для использования в подписях MLSAG наряду с соответствующими образами ключей. Кроме того, к одному из обязательств по псевдовыходу добавляется скалярная величина, отправленная посредником (умноженная на $G$). Пользователи создают для своих выходов частичные доказательства Части A и отправляют всю эту информацию посреднику. Посредник проверяет сбалансированность сумм во входах и выходах и отправляет полный список частичных доказательств Части A каждому участнику.

3. Каждый участник вычисляет совокупный запрос A и генерирует частичные доказательства Части B, которые отправляются посреднику. Посредник собирает частичные доказательства и распределяет их среди всех остальных участников.

4. Каждый участник вычисляет совокупный запрос B и генерирует частичные доказательства Части C, которые отправляются посреднику. Посредник собирает их и применяет метод логарифмического внутреннего произведения, чтобы получить окончательное доказательство. Если доказательство проходит проверку должным образом, он генерирует случайный фальшивый публичный ключ «базовой» транзакции, снижающий риск атаки Януса, и отправляет каждому участнику сообщение для подписания MLSAG.

5. Каждый участник завершает подписание MLSAG и отправляет всё посреднику. Как только посредник получит все части, он сможет завершить построение транзакции и отправить её для включения в блокчейн. Он также может отправить идентификатор транзакции каждому из участников, чтобы они могли подтвердить, что она была опубликована.

# Список используемой литературы


[1] Add intervening v5 fork for increased min block size (Реализация форка v5 с целью повышения минимального размера блока). https://github.com/monero-project/monero/pull/1869 [Online; accessed 05/24/2018].

[2] CoinJoin (Протокол CoinJoin). https://en.bitcoin.it/wiki/CoinJoin [Online; accessed 01/26/2020].

[3] Cryptography - What is a cryptographic oracle? (Криптография — что такое криптографический оракул?). https://security.stackexchange.com/questions/10617/what-is-a-cryptographic-oracle [Online; accessed 04/22/2018].

[4] Cryptography Tutorial (Учебник по криптографии). https://www.tutorialspoint.com/cryptography/index.htm [Online; accessed 05/19/2018].

[5] Cryptonote Address Tests (Тестирование адресов Cryptonote). https://xmr.llcoins.net/addresstests.html [Online; accessed 04/19/2018].

[6] Directed acyclic graph (Направленный ациклический граф). https://en.wikipedia.org/wiki/Directed_acyclic_graph [Online; accessed 05/27/2018].

[7] Extended Euclidean algorithm (Расширенный алгоритм Евклида). https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm [Online; accessed 03/19/2020].

[8] Hackerone #501585 (Публикация Hackerone #501585). https://hackerone.com/reports/501585 [Online; accessed 12/28/2019].

[9] How do payment ids work? (Как работают идентификаторы платежей?). https://monero.stackexchange.com/questions/1910/how-do-payment-ids-work [Online; accessed 04/21/2018].

[10] Modular arithmetic (Арифметические операции над абсолютными значениями чисел). https://en.wikipedia.org/wiki/Modular_arithmetic [Online; accessed 04/16/2018].

[11] Monero 0.13.0 “Beryllium Bullet" Release (Релиз Monero 0.13.0 “Beryllium Bullet"). https://www.getmonero.org/2018/10/11/monero-0.13.0-released.html [Online; accessed 10/12/2019].

[12] Monero 0.14.0 “Boron Butterfly" Release (Релиз Monero 0.14.0 “Boron Butterfly"). https://web.getmonero.org/2019/02/25/monero-0.14.0-released.html [Online; accessed 10/12/2019].

[13] Monero inception and history (Происхождение и история Monero). 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].

[14] Monero v0.9.3 - Hydrogen Helix - released! (Релиз Monero v0.9.3 Hydrogen Helix!). https://www.reddit.com/r/Monero/comments/4bgw4z/monero_v093_hydrogen_helix_released_urgent_and/ [Online; accessed 05/24/2018].

[15] RandomX (Алгоритм RandomX). https://www.monerooutreach.org/stories/RandomX.php [Online; accessed 10/12/2019].

[16] Ring CT; Moneropedia (Монеропедия, Протокол RingCT). https://getmonero.org/resources/moneropedia/ringCT.html [Online; accessed 06/05/2018].

[17] Tail emission (Хвостовая эмиссия). https://getmonero.org/resources/moneropedia/tail-emission.html [Online; accessed 05/24/2018].

[18] Trust the math? An Update (Верите ли вы математике? Обновлённая версия). http://www.math.columbia.edu/~woit/wordpress/?p=6522 [Online; accessed 04/04/2018].

[19] Useful for learning about Monero coin emission (Что нужно знать об эмиссии Monero?). https://www.reddit.com/r/Monero/comments/512kwh/useful_for_learning_about_monero_coin_emission/d78tpgi/ [Online; accessed 05/25/2018].

[20] What is a premine? (Что такое премайнинг?). https://www.cryptocompare.com/coins/guides/what-is-a-premine/ [Online; accessed 06/11/2018].

[21] What is the block maturity value seen in many pool interfaces? (Что такое значение зрелости блока, используемое в интерфейсах многих пулов?). https://monero.stackexchange.com/questions/2251/what-is-the-block-maturity-value-seen-in-many-pool-interfaces [Online; accessed 05/26/2018].

[22] XOR – from Wolfram Mathworld (Операция «исключающее ИЛИ» — материалы Wolfram Mathworld). http://mathworld.wolfram.com/XOR.html [Online; accessed 04/21/2018].

[23] Federal Information Processing Standards Publication (FIPS 197). Advanced Encryption Standard (AES) (Федеральные стандарты обработки информации (FIPS 197). Продвинутый стандарт шифрования (AES), 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf [Online; access 03/04/2020].

[24] The .xz File Format (Формат файлов .xz), August 2009. https://tukaani.org/xz/xz-file-format.txt section 1.2 [Online; accessed 04/02/2020].

[25] Fungible (Взаимозаменяемость), July 2014. https://wiki.mises.org/wiki/Fungible [Online; accessed 03/31/2020].

[26] Analysis of Bitcoin Transaction Size Trends (Анализ перспектив изменения размера транзакций Bitcoin), October 2015. https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends [Online; accessed 01/17/2020].

[27] NIST Releases SHA-3 Cryptographic Hash Standard (NIST публикует стандарт криптографической хеш-функции SHA-3), August 2015. https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard [Online; accessed 06/02/2018].

[28] Base58Check encoding (Base58Check кодирование), November 2017. https://en.bitcoin.it/wiki/Base58Check_encoding [Online; accessed 02/20/2020].

[29] Monero cryptonight variants, and add one for v7 (Варианты реализации протокола cryptonight Monero и добавление одного из них в версии v7), April 2018. https://github.com/monero-project/monero/pull/3253 [Online; accessed 05/23/2018].

[30] Payment Reversal Explained + 10 Ways to Avoid Them (Как происходит отмена платежа и 10 способов, как этого избежать), October 2018. https://tidalcommerce.com/learn/payment-reversal [Online; accessed 02/11/2020].

[31] Diffie–Hellman problem (Решение задачи Диффи-Хеллмана), December 2019. https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_problem [Online; accessed 03/31/2020].

[32] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - Privacy in the Blockchain (Monero — обеспечение анонимности данных блокчейна). Cryptology ePrint Archive, Report 2018/535, 2018. https://eprint.iacr.org/2018/535.

[33] Kurt M. Alonso and Koe. Zero to Monero - First Edition (От нуля к Monero - первая редакция, июнь 2018), June 2018. https://web.getmonero.org/library/Zero-to-Monero-1-0-0.pdf [Online; accessed 01/15/2020].

[34] Kristov Atlas. Kristov Atlas Security Advisory 20140609-0 (Советы по обеспечению безопасности от Кристова Атласа 20140609-0), June 2014. https://www.coinjoinsudoku.com/advisory/ [Online; accessed 01/26/2020].

[35] Adam Back. Ring signature efficiency (Эффективность кольцевых подписей). BitcoinTalk, 2015. https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684 [Online; accessed 04/04/2018].

[36] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records (Curve25519: новые рекорды скорости работы протокола Диффи-Хеллмана). 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. https://cr.yp.to/ecdh/curve25519-20060209.pdf [Online; accessed 03/04/2020].

[37] Daniel J. Bernstein. ChaCha, a variant of Salsa20 (ChaCha, вариант Salsa20), January 2008. https://cr.yp.to/chacha/chacha-20080120.pdf [Online; accessed 03/04/2020].

[38] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. *Twisted Edwards Curves (Скрученные эллиптические кривые Эдвардса)*, pages 389–405. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. https://eprint.iacr.org/2008/013.pdf [Online; accessed 02/13/2020].

[39] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures (Высокоскоростное создание предельно безопасных подписей). *Journal of Cryptographic Engineering*, 2(2):77–89, Sep 2012. https://ed25519.cr.yp.to/ed25519-20110705.pdf [Online; accessed 03/04/2020].

[40] Daniel J. Bernstein and Tanja Lange. *Faster Addition and Doubling on Elliptic Curves (Быстрое сложение и удвоение на эллиптических кривых)*, pages 29–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. https://eprint.iacr.org/2007/286.pdf [Online; accessed 03/04/2020].

[41] Karina Bjørnholdt. Dansk politi har knækket bitcoin-koden (Датская полиция взломала код Bitcoin), May 2017. http://www.dansk-politi.dk/artikler/2017/maj/dansk-politi-har-knaekket-bitcoin-koden [Online; accessed 04/04/2018].

[42] Dan Boneh and Victor Shoup. A Graduate Course in Applied Cryptography (Основной учебный курс по прикладной криптографии). https://crypto.stanford.edu/~dabo/cryptobook/ [Online; accessed 12/30/2019].

[43] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More (Bulletproofs: короткие доказательства для построения конфиденциальных транзакций и многого другого). https://eprint.iacr.org/2017/1066.pdf [Online; accessed 10/28/2018].

[44] Francisco Cabañas. Francisco Cabanas - Critical Role of Min Block Reward Trail Emission - DEF CON 27 Monero Village (Франсиско Кабаньяс — Критическая роль минимального вознаграждения за вычисление блока при хвостовой эмиссии), December 2019. https://www.youtube.com/watch?v=IlghysBBuyU [Online; accessed 01/15/2020].

[45] Francisco Cabañas. Lightning talk: An Overview of Monero's Adaptive Blockweight Approach to Scaling (Краткий доклад на тему адаптивного подхода Monero к весу блоков в целях улучшения масштабирования), December 2019. https://frab.riat.at/en/36C3/public/events/125.html [Online; accessed 01/12/2020].

[46] Francisco Cabañas. MoneroKon 2019 - Spam Mitigation and Size Control in Permissionless Blockchains (Выступление на конференции MoneroKon 2019 — Подавление спама и контроль размера не требующих разрешения блокчейнов), June 2019. https://www.youtube.com/watch?v=Hbm0ub3qWw4&list=LL2HXH-vq_sTPXMNP4fZNKRw&index=10&t=0s [Online; accessed 01/10/2020].

[47] Miles Carlsten, Harry Kalodner, Matthew Weinberg, and Arvind Narayanan. On the Instability of Bitcoin Without the Block Reward (По вопросу нестабильности Bitcoin при отсутствии вознаграждения за вычисление блока), 2016. http://randomwalker.info/publications/mining_CCS.pdf [Online; accessed 01/12/2020].

[48] Chainalysis. THE 2020 STATE OF CRYPTO CRIME (Ситуация с крипто-преступлениями в 2020 г.), January 2020. https://go.chainalysis.com/rs/503-FAP-074/images/2020-Crypto-Crime-Report.pdf [Online; accessed 02/11/2020].

[49] David Chaum and Eugène Van Heyst. Group Signatures (Групповые подписи). 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. https://chaum.com/publications/Group_Signatures.pdf [Online; accessed 03/04/2020].

[50] dalek cryptography. Bulletproofs (Доказательства Bulletproofs). https://doc-internal.dalek.rs/bulletproofs/index.html [Online; accessed 03/02/2020].

[51] Michael Davidson and Tyler Diamond. On the Profitability of Selfish Mining Against Multiple Difficulty Adjustment Algorithms (По вопросу прибыльности «эгоистичного» майнинга при реализации алгоритмов многократной корректировки сложности). Cryptology ePrint Archive, Report 2020/094, 2020. https://eprint.iacr.org/2020/094 [Online; accessed 03/26/2020].

[52] W. Diffie and M. Hellman. New Directions in Cryptography (Новые направления в криптографии). *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006. https://ee.stanford.edu/~hellman/publications/24.pdf [Online; accessed 03/04/2020].

[53] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the Security of Two-Round Multi-Signatures (По вопросу безопасности создаваемых в два этапа мультиподписей). Cryptology ePrint Archive, Report 2018/417, 2018. https://eprint.iacr.org/2018/417 [Online; accessed 02/07/2020].

[54] Justin Ehrenhofer. Justin Ehrenhofer - Improving Monero Release Schedule - DEF CON 27 Monero Village (Джастин Эренхофер — Улучшение графика эмиссии Monero (выступление на конференции DEF CON 27)), December 2019. https://www.youtube.com/watch?v=MjNXmJUk2Jo timestamp 22:15 [Online; accessed 03/20/2020].

[55] Justin Ehrenhofer. Monero Adds Blockchain Pruning and Improves Transaction Efficiency (Monero реализует возможность обрезания блокчейна и повышает эффективность транзакций), February 2019. https://web.getmonero.org/2019/02/01/pruning.html [Online; accessed 12/30/2019].

[56] Justin Ehrenhofer and knacc. Advisory note for users making use of subaddresses (Рекомендация для пользователей, использующих подадреса), October 2019. https://web.getmonero.org/2019/10/18/subaddress-janus.html [Online; accessed 01/02/2020].

[57] Serhack et al. Mastering Monero (Овладевая Monero), December 2019. https://masteringmonero.com/ [Online; accessed 01/10/2020].

[58] Exantech. Methods of anonymous blockchain analysis: an overview (Методы анализа анонимных блокчейнов - обзор), November 2019. https://medium.com/@exantech/methods-of-anonymous-blockchain-analysis-an-overview-d700e27ea98c [Online; accessed 01/26/2020].

[59] Ittay Eyal and Emin Gün Sirer. Majority is not Enough: Bitcoin Mining is Vulnerable (Большинство — это ещё не все: майнинг Bitcoin небезопасен). *CoRR*, abs/1311.0243, 2013. https://arxiv.org/pdf/1311.0243.pdf [Online; accessed 03/04/2020].

[60] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems (Как доказать, что вы — это вы: практические решения проблем, связанных с идентификацией и подписанием). In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-47721-7_12.pdf [Online; accessed 03/04/2020].

[61] Ryo "fireice_uk" Cryptocurrency. On-chain tracking of Monero and other Cryptonotes (Отслеживание данных блокчейна Monero и других криптовалют на базе Cryptonote), April 2019. https://medium.com/@crypto_ryo/on-chain-tracking-of-monero-and-other-cryptonotes-e0afc6752527 [Online; accessed 03/25/2020].

[62] Riccardo “fluffypony" Spagni. Monero 0.12.0.0 “Lithium Luna" Release (Релиз Monero 0.12.0.0 “Lithium Luna"), March 2018. https://web.getmonero.org/2018/03/29/monero-0.12.0.0-released.html [Online; accessed 02/18/2020].

[63] Riccardo “fluffypony" Spagni and luigi1111. Disclosure of a Major Bug in Cryptonote Based Currencies (Обнаружение главного бага в криптовалютах на базе Cryptonote), May 2017. https://getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html [Online; accessed 04/10/2018].

[64] David Friedman. A Positive Account of Property Rights (Очко в пользу прав собственности), 1994. http://www.daviddfriedman.com/Academic/Property/Property.html [Online; accessed 03/18/2020].

[65] Eiichiro Fujisaki and Koutarou Suzuki. *Traceable Ring Signature (Отслеживаемая кольцевая подпись)*, pages 181–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. https://link.springer.com/content/pdf/10.1007%2F978-3-540-71677-8_13.pdf [Online; accessed 03/04/2020].

[66] glv2. Varint description (Описание варинта); Issue #2340. https://github.com/monero-project/monero/issues/2340#issuecomment-324692291 [Online; accessed 06/14/2018].

[67] Brandon Goodell, Sarang Noether, and Arthur Blue. Concise linkable ring signatures and applications (Компактные связываемые кольцевые подписи), MRL-0011, September 2019. https://web.getmonero.org/resources/research-lab/pubs/MRL-0011.pdf [Online; accessed 02/02/2020].

[68] Thomas C Hales. The NSA back door to NIST (Лазейка АНБ в NIST). *Notices of the AMS*, 61(2):190–192. https://www.ams.org/notices/201402/rnoti-p190.pdf [Online; accessed 03/04/2020].

[69] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography (Руководство по криптографии на эллиптических кривых)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[70] Howard “hyc" Chu. RandomX (Алгоритм RandomX), Pull Request #5549, May 2019. https://github.com/monero-project/monero/pull/5549 [Online; accessed 03/03/2020].

[71] Aram Jivanyan. Lelantus: Towards Confidentiality and Anonymity of Blockchain Transactions from Standard Assumptions (Lelantus: шаг в сторону анонимности и конфиденциальности транзакций в блокчейне со стандартными допусками). Cryptology ePrint Archive, Report 2019/373, 2019. https://eprint.iacr.org/2019/373.pdf [Online; accessed 03/04/2020].

[72] Don Johnson and Alfred Menezes. The Elliptic Curve Digital Signature Algorithm (ECDSA) (Алгоритм цифровой подписи на эллиптической кривой (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].

[73] JollyMort. Monero Dynamic Block Size and Dynamic Minimum Fee (Динамический размер блока и динамический размер минимальной комиссии Monero), March 2017. https://github.com/JollyMort/monero-research/blob/master/Monero%20Dynamic%20Block%20Size%20and%20Dynamic%20Minimum%20Fee/Monero%20Dynamic%20Block%20Size%20and%20Dynamic%20Minimum%20Fee%20-%20DRAFT.md [Online; accessed 01/12/2020].

[74] S. Josefsson, SJD AB, and N. Moeller. EdDSA and Ed25519 (EdDSA и Ed25519). Internet Research Task Force (IRTF), 2015. https://tools.ietf.org/html/draft-josefsson-eddsa-ed25519-03 [Online; accessed 05/11/2018].

[75] Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA) (Алгоритм цифровой подписи на эллиптической кривой Эдвардса EdDSA). RFC 8032, January 2017. https://rfc-editor.org/rfc/rfc8032.txt [Online; accessed 03/04/2020].

[76] kenshi84. Subaddresses (Подадреса), Pull Request #2056, May 2017. https://github.com/monero-project/monero/pull/2056 [Online; accessed 02/16/2020].

[77] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal Security Proofs for Signatures from Identification Schemes (Оптимальные доказательства безопасности подписей на базе схем идентификации). 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. https://eprint.iacr.org/2016/191.pdf [Online; accessed 03/04/2020].

[78] Bradley Kjell. Big Endian and Little Endian (Форматы с порядком следования байтов со старшего и с младшего). https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html [Online; accessed 01/23/2020].

[79] Alexander Klimov. ECC Patents? (Патенты ECC?), October 2005. http://article.gmane.org/gmane.comp.encryption.general/7522 [Online; accessed 04/04/2018].

[80] koe and jtgrassie. Historical significance of FEE_PER_KB_OLD (Историческая важность расчета комиссии по килобайтам), December 2019. https://monero.stackexchange.com/questions/11864/historical-significance-of-fee-per-kb-old [Online; accessed 01/02/2020].

[81] koe and jtgrassie. Complete extra field structure (standard interpretation) (Полная структура дополнительного поля — стандартная интерпретация), January 2020. https://monero.stackexchange.com/questions/11888/complete-extra-field-structure-standard-interpretation [Online; accessed 01/05/2020].

[82] Mitchell Krawiec-Thayer. MoneroKon 2019 - Visualizing Monero: A Figure is Worth a Thousand Logs (Выступление на конференции MoneroKon 2019 — Визуализация Monero: один рисунок лучше тысячи логов), June 2019. https://www.youtube.com/watch?v=XIrqyxU3k5Q [Online; accessed 01/06/2020].

[83] Mitchell Krawiec-Thayer. Numerical simulation for upper bound on dynamic blocksize expansion (Численное моделирование верхней границы при динамическом увеличении размера блока), January 2019. https://github.com/noncesense-research-lab/Blockchain_big_bang/blob/master/models/Isthmus_Bx_big_bang_model.ipynb [Online; accessed 01/08/2020].

[84] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Thyagarajan, and Jiafan Wang. Omniring: Scaling Private Payments Without Trusted Setup (Omniring: Масштабирование анонимных платежей без доверенных настроек). pages 31–48, 11 2019. https://eprint.iacr.org/2019/580.pdf [Online; accessed 03/04/2020].

[85] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. *Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups (Связываемые подписи спонтанной анонимной группы для специально создаваемых групп)*, pages 325–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. https://eprint.iacr.org/2004/027.pdf [Online; accessed 03/04/2020].

[86] Jan Macheta, Sarang Noether, Surae Noether, and Javier Smooth. Counterfeiting via Merkle Tree Exploits within Virtual Currencies Employing the CryptoNote Protocol (Подделка виртуальных валют на базе протокола Cryptonote с использованием эксплойтов дерева Меркла), MRL-0002, September 2014. https://web.getmonero.org/resources/research-lab/pubs/MRL-0002.pdf [Online; accessed 05/27/2018].

[87] Ueli Maurer. Unifying Zero-Knowledge Proofs of Knowledge (Унификация доказательств знания с нулевым разглашением). In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, pages 272–286, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. https://www.crypto.ethz.ch/publications/files/Maurer09.pdf [Online; accessed 03/04/2020].

[88] Greg Maxwell. Confidential Transactions (Конфиденциальные транзакции). https://people.xiph.org/~greg/confidential_values.txt [Online; accessed 04/04/2018].

[89] Gregory Maxwell and Andrew Poelstra. Borromean Ring Signatures (Кольцевые подписи Борромео). 2015. https://pdfs.semanticscholar.org/4160/470c7f6cf05ffc81a98e8fd67fb0c84836ea.pdf [Online; accessed 04/04/2018].

[90] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr Multi-Signatures with Applications to Bitcoin (Применение простых мультиподписей Шнорра в Bitcoin). May 2018. https://eprint.iacr.org/2018/068.pdf [Online; accessed 03/01/2020].

[91] Sarah Meiklejohn and Claudio Orlandi. Privacy-Enhancing Overlays in Bitcoin (Применение повышающих уровень анонимности оверлеев в Bitcoin). https://fc15.ifca.ai/preproceedings/bitcoin/paper_5.pdf [Online; accessed 01/26/2020].

[92] R. C. Merkle. Protocols for Public Key Cryptosystems (Протоколы криптосистем на базе публичных ключей). In *1980 IEEE Symposium on Security and Privacy*, pages 122–122, April 1980. http://www.merkle.com/papers/Protocols.pdf [Online; accessed 03/04/2020].

[93] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. An Empirical Analysis of Linkability in the Monero Blockchain (Эмпирический анализ связываемости в блокчейне Monero). *CoRR*, abs/1704.04299, 2017. https://arxiv.org/pdf/1704.04299.pdf [Online; accessed 03/04/2020.

[94] Victor S Miller. Use of Elliptic Curves in Cryptography (Использование эллиптических кривых в криптографии). In *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, Berlin, Heidelberg, 1986. Springer-Verlag. https://link.springer.com/content/pdf/10.1007/3-540-39799-X_31.pdf [Online; accessed 03/04/2020].

[95] Nicola Minichiello. The Bitcoin Big Bang: Tracking Tainted Bitcoins (Большой взрыв Bitcoin: отслеживание помеченных монет), June 2015. https://bravenewcoin.com/insights/the-bitcoin-big-bang-tracking-tainted-bitcoins [Online; accessed 03/31/2020].

[96] Ludwig Von Mises. Human Action: A Treatise on Economics; The Scholar's Edition (Человеческие действия: монография по экономике. Издание для студентов), 1949. https://cdn.mises.org/Human%20Action_3.pdf [Online; accessed 03/05/2020].

[97] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System (Bitcoin: одноранговая электронная денежная система), 2008. http://bitcoin.org/bitcoin.pdf [Online; accessed 03/04/2020].

[98] Arvind Narayanan. Bitcoin is unstable without the block reward (Bitcoin будет нестабилен без вознаграждений за вычисление блоков), October 2016. https://freedom-to-tinker.com/2016/10/21/bitcoin-is-unstable-without-the-block-reward/ [Online; accessed 01/12/2020].

[99] Arvind Narayanan and Malte Möser. Obfuscation in Bitcoin: Techniques and Politics (Обфускация в Bitcoin: технологии и политика). *CoRR*, abs/1706.05432, 2017. https://arxiv.org/ftp/arxiv/papers/1706/1706.05432.pdf [Online; accessed 03/04/2020].

[100] Y. Nir, Check Point, A. Langley, and Google Inc. ChaCha20 and Poly1305 for IETF Protocols (Применение ChaCha20 и Poly1305 в протоколах IETF). Internet Research Task Force (IRTF), May 2015. https://tools.ietf.org/html/rfc7539 [Online; accessed 05/11/2018].

[101] Sarang Noether. Janus mitigation (Как избежать атаки Януса), Issue #62, January 2020. https://github.com/monero-project/research-lab/issues/62 [Online; accessed 02/17/2020].

[102] Sarang Noether. Multisignature implementation (Реализация мультиподписей), Issue #67, January 2020. https://github.com/monero-project/research-lab/issues/67 [Online; accessed 02/16/2020].

[103] Sarang Noether. Transaction proofs (InProofV1 and OutProofV1) have incomplete Schnorr challenges (Доказательства транзакций (InProofV1 и OutProofV1) имеют неполные запросы Шнорра), Issue #60, January 2020. https://github.com/monero-project/research-lab/issues/60 [Online; accessed 02/20/2020].

[104] Sarang Noether. WIP: Updated transaction proofs and tests (WIP: обновлённые доказательства транзакций и тесты), Pull Request #6329, February 2020. https://github.com/monero-project/monero/pull/6329 [Online; accessed 02/20/2020].

[105] Sarang Noether and Brandon Goodell. An efficient implementation of Monero subaddresses (Эффективная реализация подадресов Monero), MRL-0006, October 2017. https://web.getmonero.org/resources/research-lab/pubs/MRL-0006.pdf [Online; accessed 04/04/2018].

[106] Sarang Noether and Brandon Goodell. Triptych: logarithmic-sized linkable ring signatures with applications (Triptych: логарифмически масштабируемые связываемые кольцевые подписи и их применение). Cryptology ePrint Archive, Report 2020/018, 2020. https://eprint.iacr.org/2020/018.pdf [Online; accessed 03/04/2020.

[107] Shen Noether. Understanding ge_fromfe_frombytes_vartime (Как работает ge_fromfe_frombytes_vartime). https://web.getmonero.org/resources/research-lab/pubs/ge_fromfe.pdf [Online; accessed 01/20/2020].

[108] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring Confidential Transactions (Кольцевые конфиденциальные транзакции), MRL-0005, February 2016. https://web.getmonero.org/resources/research-lab/pubs/MRL-0005.pdf [Online; accessed 06/15/2018].

[109] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring Multisignature (Кольцевая мультиподпись), April 2016. https://web.archive.org/web/20161023010706/https://shnoe.files.wordpress.com/2016/03/mrl-0008_april28.pdf [Online; accessed via WayBack Machine 01/21/2020].

[110] Shen Noether and Sarang Noether. Monero is Not That Mysterious (Monero не так загадочна, как кажется), MRL-0003, September 2014. https://web.getmonero.org/resources/research-lab/pubs/MRL-0003.pdf [Online; accessed 06/15/2018].

[111] Shen Noether and Sarang Noether. Thring Signatures and their Applications to Spender-Ambiguous Digital Currencies (Thring-подписи и их применение в цифровых валютах, скрывающих отправителя), MRL-0009, November 2018. https://web.getmonero.org/resources/research-lab/pubs/MRL-0009.pdf [Online; accessed 01/15/2020].

[112] Michael Padilla. Beating Bitcoin bad guys (Побеждая плохих парней из Bitcoin), August 2016. http://www.sandia.gov/news/publications/labnews/articles/2016/19-08/bitcoin.html [Online; accessed 04/04/2018].

[113] Torben Pryds Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing (Неинтерактивное и информационно-теоретическое совместное использование верифицируемых секретов)*, pages 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992. https://www.cs.cornell.edu/courses/cs754/2001fa/129.PDF [Online; accessed 03/04/2020].

[114] rbrunner7. 2/2 Multisig in CLI Wallet (Использование схемы multisig «2 из 2» в CLI-кошельке), January 2018. https://taiga.getmonero.org/project/rbrunner7-really-simple-multisig-transactions/wiki/22-multisig-in-cli-wallet [Online; accessed 01/21/2020].

[115] René "rbrunner7" Brunner. Multisig transactions with MMS and CLI wallet (Проведение multisig-транзакций в MMS и CLI кошельках). https://web.getmonero.org/resources/user-guides/multisig-messaging-system.html [Online; accessed 01/21/2020].

[116] René "rbrunner7" Brunner. Project Rationale From the Initiator (Обоснование проекта инициатором), January 2018. https://taiga.getmonero.org/project/rbrunner7-really-simple-multisig-transactions/wiki/home [Online; accessed 01/21/2020].

[117] René "rbrunner7" Brunner. Basic Monero support ready for assessment (Базовая поддержка Monero, подлежащая оценке), Issue #1638, June 2019. https://github.com/OpenBazaar/openbazaar-go/issues/1638 [Online; accessed 02/11/2020].

[118] Jamie Redman. Industry Execs Claim Freshly Minted 'Virgin Bitcoins' Fetch 20% Premium (Руководящие представители отрасли заявляют, что недавно созданные Virgin Bitcoin приносят 20% прибыли), March 2020. https://news.bitcoin.com/industry-execs-freshly-minted-virgin-bitcoins/ [Online; accessed 03/31/2020].

[119] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret (Как раскрыть секрет). 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].

[120] SafeCurves. SafeCurves: choosing safe curves for elliptic-curve cryptography (SafeCurves: выбор безопасных кривых при использовании криптографии на эллиптических кривых), 2013. https://safecurves.cr.yp.to/rigid.html [Online; accessed 03/25/2020].

[121] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards (Эффективная идентификация и использование подписей со смарт-картами). In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York. https://link.springer.com/content/pdf/10.1007%2F0-387-34805-0_22.pdf [Online; accessed 03/04/2020].

[122] Paola Scozzafava. Uniform distribution and sum modulo m of independent random variables (Единообразное распределение и суммирование по модулю m независимых случайных переменных. Статистика и вероятность). *Statistics & Probability Letters*, 18(4):313 – 314, 1993. https://sci-hub.tw/https://www.sciencedirect.com/science/article/abs/pii/016771529390021A [Online; accessed 03/04/2020].

[123] Bassam El Khoury Seguias. Monero Building Blocks (Строительные блоки Monero), 2018. https://delfr.com/category/monero/ [Online; accessed 10/28/2018].

[124] Seigen, Max Jameson, Tuomo Nieminen, Neocortex, and Antonio M. Juarez. CryptoNight Hash Function (Хеш-функция CryptoNight). CryptoNote, March 2013. https://cryptonote.org/cns/cns008.txt [Online; accessed 04/04/2018].

[125] QingChun ShenTu and Jianping Yu. Research on Anonymization and De-anonymization in the Bitcoin System (Исследование возможностей анонимизации и деанонимизации в системах Bitcoin). *CoRR*, abs/1510.07782, 2015. https://arxiv.org/ftp/arxiv/papers/1510/1510.07782.pdf [Online; accessed 03/04/2020].

[126] stoffu. Reserve proof (Доказательство резервов), Pull Request #3027, December 2017. https://github.com/monero-project/monero/pull/3027 [Online; accessed 02/24/2020].

[127] thankful_for_today. [ANN][BMR] Bitmonero - a new coin based on CryptoNote technology - LAUNCHED (Запуск новой монеты на базе технологии CryptoNote — [ANN][BMR] Bitmonero), April 2014. Monero's actual launch date was April 18$^{th}$, 2014. https://bitcointalk.org/index.php?topic=563821.0 [Online; accessed 05/24/2018].

[128] Manny Trillo. Visa Transactions Hit Peak on Dec. 23 (Количество транзакций, совершаемых с использованием Visa, достигло своего пика 23 декабря), January 2011. https://www.visa.com/blogarchives/us/2011/01/12/visa-transactions-hit-peak-on-dec-23/index.html [Online; accessed 01/10/2020].

[129] Alicia Tuovila. Audit (Аудит), July 2019. https://www.investopedia.com/terms/a/audit.asp [Online; accessed 02/24/2020].

[130] UkoeHB. Proof an output has not been spent (Доказательство того, что выход не был потрачен), Issue #68, January 2020. https://github.com/monero-project/research-lab/issues/68 [Online; accessed 02/19/2020].

[131] UkoeHB. Reduce minimum fee variability (Снижение частоты изменения минимальной комиссии), Issue #70, February 2020. https://github.com/monero-project/research-lab/issues/70 [Online; accessed 03/20/2020].

[132] UkoeHB. Treat pre-RingCT outputs like coinbase outputs (Рассмотрение выходов, созданных до реализации RingCT, в качестве coinbase-выходов), Issue #59, January 2020. https://github.com/monero-project/research-lab/issues/59 [Online; accessed 03/22/2020].

[133] Maciej Ulas. Rational points on certain hyperelliptic curves over finite fields (Рациональные точки на некоторых гиперэллиптических кривых в конечных полях), 2007. https://arxiv.org/pdf/0706.1448.pdf [Online; accessed 03/03/2020].

[134] user36100 and cooldude45. Which entities are related to Bytecoin and Minergate? (Что связано с Bytecoin и Minergate?), December 2016. https://monero.stackexchange.com/questions/2930/which-entities-are-related-to-bytecoin-and-minergate [Online; accessed 01/17/2020].

[135] user36303 and koe. Duplicate ring members (Дублирование участников кольца), January 2020. https://monero.stackexchange.com/questions/11882/duplicate-ring-members [Online; accessed 01/18/2020].

[136] Nicolas van Saberhagen. CryptoNote V2.0. https://cryptonote.org/whitepaper.pdf [Online; accessed 04/04/2018].

[137] David Wagner. A Generalized Birthday Problem (Обобщение парадокса дней рождения). In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 288–304, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-45708-9_19.pdf [Online; accessed 02/07/2020].

[138] Adam “waxwing" Gibson. From Zero (Knowledge) To Bulletproofs (От доказательств с нулевым разглашением до Bulletproofs), March 2018. https://github.com/AdamISZ/from0k2bp/blob/master/from0k2bp.pdf [Online; accessed 03/01/2020].

[139] Adam “waxwing" Gibson. Avoiding Wagnerian Tragedies (Как избежать вагнерианских трагедий), December 2019. https://joinmarket.me/blog/blog/avoiding-wagnerian-tragedies/ [Online; accessed 03/01/2020].

[140] Albert Werner, Montag, Prometheus, and Tereno. CryptoNote Transaction Extra Field (Дополнительное поле в транзакциях Cryptonote). CryptoNote, October 2012. https://cryptonote.org/cns/cns005.txt [Online; accessed 04/04/2018].

[141] Ursula Whitcher. The Birthday Problem (Парадокс дней рождения). http://mathforum.org/dr.math/faq/faq.birthdayprob.html [Online; accessed 02/07/2020].

[142] Wikibooks. Cryptography/Prime Curve/Standard Projective Coordinates (Криптография/Простая кривая/Стандартные проективные координаты), March 2011. https://en.wikibooks.org/wiki/Cryptography/Prime_Curve/Standard_Projective_Coordinates [Online; accessed 03/03/2020].

[143] Tsz Hon Yuen, Shi-feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. RingCT 3.0 for Blockchain Confidential Transaction: Shorter Size and Stronger Security (Версия RingCT 3.0 с возможностью реализации конфиденциальных транзакций в блокчейне: уменьшение размера и повышение уровня безопасности). Cryptology ePrint Archive, Report 2019/508, 2019. https://eprint.iacr.org/2019/508.pdf [Online; accessed 03/04/2020].

[144] zawy12. Summary of Difficulty Algorithms (Краткое описание алгоритмов определения сложности), Issue #50, December 2019. https://github.com/zawy12/difficulty-algorithms/issues/50 [Online; accessed 02/11/2020].

# Приложения

# ПРИЛОЖЕНИЕ A

## Структура транзакций `RCTTypeBulletproof2`

В этом приложении мы приводим пример распечатанных данных реальной транзакции Monero типа `RCTTypeBulletproof2`, а также свои пояснения к соответствующим полям.

Данные были получены при помощи блок-эксплорера `https://xmrchain.net`. Также их можно извлечь, воспользовавшись командой print tx `print_tx <TransactionID> +hex +json` в демон-программе `monerod` в неотключённом режиме. `<TransactionID>` является хешем транзакции (см. подпункт 7.4.1). Первая распечатанная строка показывает фактический запуск команды.

Чтобы воспроизвести наши результаты, пользователи могут сделать следующее:

1. Вам понадобится инструмент с поддержкой командной строки (CLI) Monero, который можно скачать на `https://web.getmonero.org/downloads/` (и не только). Следует выбрать Command Line Tools Only (только инструменты с поддержкой командной строки) для вашей операционной системы, переместить файл в нужное место и разархивировать его.

2. После этого следует открыть terminal/command line (командная строка) и перейти к папке, созданной в результате разархивирования.

3. Используя `monerod`, открыть демон-программу `./monerod`. Начнётся загрузка блокчейна Monero. К сожалению, на данный момент простого способа распечатки транзакций на вашей системе не существует (например, без использования блокчейн-эксплорера), поэтому приходится загружать блокчейн.

4. После того как процесс синхронизации будет завершён, можно будет использовать такие команды, как `print_tx`. Чтобы узнать другие команды, воспользуйтесь разделом `help` (помощь).

Компонент `rctsig_prunable`, как следует из его названия, теоретически можно *вырезать* из блокчейна. То есть после того, как блок будет согласован, а существующие правила, определяющие длину блокчейна, исключают всяческую возможность атаки путём двойной траты, всё это поле можно будет вырезать и заменить его хешем в дереве Меркла.

Образы ключей хранятся отдельно в той части транзакций, которая не может быть удалена. Эти компоненты важны с точки зрения обнаружения атак путём двойной траты и не могут быть вырезаны.

В нашем примере транзакция имеет 2 входа и 2 выхода и была добавлена в блокчейн с временной меткой 2020-03-02 19:01:10 UTC (как было указано майнером блока).

```
print_tx 84799c2fc4c18188102041a74cef79486181df96478b717e8703512c7f7f3349
Found in blockchain at height 2045821
{
  "version": 2,
  "unlock_time": 0,
  "vin": [ {
      "key": {
        "amount": 0,
        "key_offsets": [ 14401866, 142824, 615514, 18703, 5949, 22840, 5572, 16439,
        983, 4050, 320
        ],
        "k_image": "c439b9f0da76ca0bb17920ca1f1f3f1d216090751752b091bef9006918cb3db4"
      }
    }, {
      "key": {
        "amount": 0,
        "key_offsets": [ 14515357, 640505, 8794, 1246, 20300, 18577, 17108, 9824, 581,
        637, 1023
        ],
        "k_image": "03750c4b23e5be486e62608443151fa63992236910c41fa0c4a0a938bc6f5a37"
      }
    }
  ],
  "vout": [ {
      "amount": 0,
      "target": {
```

```
        "key": "d890ba9ebfa1b44d0bd945126ad29a29d8975e7247189e5076c19fa7e3a8cb00"
      }
    }, {
      "amount": 0,
      "target": {
        "key": "dbec330f8a67124860a9bfb86b66db18854986bd540e710365ad6079c8a1c7b0"
      }
    }
  ],
  "extra": [ 1, 3, 39, 58, 185, 169, 82, 229, 226, 22, 101, 230, 254, 20, 143,
  37, 139, 28, 114, 77, 160, 229, 250, 107, 73, 105, 64, 208, 154, 182, 158, 200,
  73, 2, 9, 1, 12, 76, 161, 40, 250, 50, 135, 231
  ],
  "rct_signatures": {
    "type": 4,
    "txnFee": 32460000,
    "ecdhInfo": [ {
        "amount": "171f967524e29632"
      }, {
        "amount": "5c2a1a9f54ccf40b"
      }],
    "outPk": [ "fed8aded6914f789b63c37f9d2eb5ee77149e1aa4700a482aea53f82177b3b41",
    "670e086e40511a279e0e4be89c9417b4767251c5a68b4fc3deb80fdae7269c17"]
  },
  "rctsig_prunable": {
    "nbp": 1,
    "bp": [ {
        "A": "98e5f23484e97bb5b2d453505db79caadf20dc2b69dd3f2b3dbf2a53ca280216",
        "S": "b791d4bc6a4d71de5a79673ed4a5487a184122321ede0b7341bc3fdc0915a796",
        "T1": "5d58cfa9b69ecdb2375647729e34e24ce5eb996b5275aa93f9871259f3a1aecd",
        "T2": "1101994fea209b71a2aa25586e429c4c0f440067e2b197469aa1a9a1512f84b7",
        "taux": "b0ad39da006404ccacee7f6d4658cf17e0f42419c284bdca03c0250303706c03",
        "mu": "cacd7ca5404afa28e7c39918d9f80b7fe5e572a92a10696186d029b4923fa200",
        "L": [ "d06404fc35a60c6c47a04e2e43435cb030267134847f7a49831a61f82307fc32",
        "c9a5932468839ee0cda1aa2815f156746d4dce79dab3013f4c9946fce6b69eff",
        "efdae043dcedb79512581480d80871c51e063fe9b7a5451829f7a7824bcc5a0b",
        "56fd2e74ac6e1766cfd56c8303a90c68165a6b0855fae1d5b51a2e035f333a1d",
        "81736ed768f57e7f8d440b4b18228d348dce1eca68f969e75fab458f44174c99",
        "695711950e076f54cf24ad4408d309c1873d0f4bf40c449ef28d577ba74dd86d",
        "4dc3147619a6c9401fec004652df290800069b776fe31b3c5cf98f64eb13ef2c"
        ],
        "R": [ "7650b8da45c705496c26136b4c1104a8da601ea761df8bba07f1249495d8f1ce",
```

```
          "e87789fbe99a44554871fcf811723ee350cba40276ad5f1696a62d91a4363fa6",
          "ebf663fe9bb580f0154d52ce2a6dae544e7f6fb2d3808531b0b0749f5152ddbf",
          "5a4152682a1e812b196a265a6ba02e3647a6bd456b7987adff288c5b0b556ec6",
          "dc0dcb2e696e11e4b62c20b6bfcb6182290748c5de254d64bf7f9e3c38fb46c9",
          "101e2271ced03b229b88228d74b36088b40c88f26db8b1f9935b85fb3ab96043",
          "b14aae1d35c9b176ac526c23f31b044559da75cf95bc640d1005bfcc6367040b"
          ],
          "a": "4809857de0bd6becdb64b85e9dfbf6085743a8496006b72ceb81e01080965003",
          "b": "791d8dc3a4ddde5ba2416546127eb194918839ced3dea7399f9c36a17f36150e",
          "t": "aace86a7a1cbdec3691859fa07fdc83eed9ca84b8a064ca3f0149e7d6b721c03"
        }
      ],
      "MGs": [ {
          "ss": [[ "d7a9b87cfa74ad5322eedd1bff4c4dca08bcff6f8578a29a8bc4ad6789dee106",
          "f08e5dfade29d2e60e981cb561d749ea96ddc7e6855f76f9b807842d1a17fe00"],
          ["de0a86d12be2426f605a5183446e3323275fe744f52fb439041ad2d59136ea0b",
          "0028f97976630406e12c54094cbbe23a23fe5098f43bcae37339bfc0c4c74c07"],
          ["f6eef1f99e605372cb1ec2b3dd4c6e56a550fec071c8b1d830b9fda34de5cc05",
          "cd98fc987374a0ac993edf4c9af0a6f2d5b054f2af601b612ea118f405303306"],
          ["5a8437575dae7e2183a1c620efbce655f3d6dc31e64c96276f04976243461e08",
          "5090103f7f73a33024fbda999cd841b99b87c45fa32c4097cdc222fa3d7e9502"],
          ["88d34246afbccbd24d2af2ba29d835813634e619912ea4ca194a32281ac14e0e",
          "eacdf59478f132dd8dbb9580546f96de194092558ffceeff410ee9eb30ce570e"],
          ["571dab8557921bbae30bda9b7e613c8a0cff378d1ec6413f59e4972f30f2470d",
          "5ca78da9a129619299304d9b03186233370023debfdaddcd49c1a338c1f0c50d"],
          ["ac8dbe6bb28839cf98f02908bd1451742a10c713fdd51319f2d42a58bf1d7507",
          "7347bf16cba5ee6a6f2d4f6a59d1ed0c1a43060c3a235531e7f1a75cd8c8530d"],
          ["b8876bd3a5766150f0fbc675ba9c774c2851c04afc4de0b17d3ac4b6de617402",
          "e39f1d2452d76521cbf02b85a6b626eeb5994f6f28ce5cf81adc0ff2b8adb907"],
          ["1309f8ead30b7be8d0c5932743b343ef6c0001cef3a4101eae98ffde53f46300",
          "370693fa86838984e9a7232bca42fd3d6c0c2119d44471d61eee5233ba53c20f"],
          ["80bc2da5fc5951f2c7406fce37a7aa72ffef9cfa21595b1b68dfab4b7b9f9f0c",
          "c37137898234f00bce00746b131790f3223f97960eefe67231eb001092f5510c"],
          ["01c89e07571fd365cac6744b34f1b44e06c1c31cbf3ee4156d08309345fdb20e",
          "a35c8786695a86c0a4e677b102197a11dadc7171dd8c2e1de90d828f050ec00f"]],
          "cc": "0d8b70c600c67714f3e9a0480f1ffc7477023c793752c1152d5df0813f75ff0f"
        }, {
          "ss": [[ "4536e585af58688b69d932ef3436947a13d2908755d1c644ca9d6a978f0f0206",
          "9aab6509f4650482529219a805ee09cd96bb439ee1766ced5d3877bf1518370b"],
          ["5849d6bf0f850fcee7acbef74bd7f02f77ecfaaa16a872f52479ebd27339760f",
          "96a9ec61486b04201313ac8687eaf281af59af9fd10cf450cb26e9dc8f1ce804"],
          ["7fe5dcc4d3eff02fca4fb4fa0a7299d212cd8cd43ec922d536f21f92c8f93f00",
```

```
111         "d306a62831b49700ae9daad44fcd00c541b959da32c4049d5bdd49be28d96701"],
112         ["2edb125a5670d30f6820c01b04b93dd8ff11f4d82d78e2379fe29d7a68d9c103",
113         "753ac25628c0dada7779c6f3f13980dfc5b7518fb5855fd0e7274e3075a3410c"],
114         ["264de632d9cb867e052f95007dfdf5a199975136c907f1d6ad73061938f49c01",
115         "dd7eb6028d0695411f647058f75c42c67660f10e265c83d024c4199bed073d01"],
116         ["b2ac07539336954f2e9b9cba298d4e1faa98e13e7039f7ae4234ac801641340f",
117         "69e130422516b82b456927b64fe02732a3f12b5ee00c7786fe2a381325bf3004"],
118         ["49ea699ca8cf2656d69020492cdfa69815fb69145e8f922bb32e358c23cebb0f",
119         "c5706f903c04c7bed9c74844f8e24521b01bc07b8dbf597621cceeeb3afc1d0c"],
120         ["a1faf85aa942ba30b9f0511141fcab3218c00953d046680d36e09c35c04be905",
121         "7b6b1b6fb23e0ee5ea43c2498ea60f4fcf62f70c7e0e905eb4d9afa1d0a18800"],
122         ["785d0993a70f1c2f0ac33c1f7632d64e34dd730d1d8a2fb0606f5770ed633506",
123         "e12777c49ffc3f6c35d27a9ccb3d9b8fed7f0864a880f7bae7399e334207280e"],
124         ["ab31972bf1d2f904d6b0bf18f4664fa2b16a1fb2644cd4e6278b63ade87b6d09",
125         "1efb04fe9a75c01a0fe291d0ae00c716e18c64199c1716a086dd6e32f63e0a07"],
126         ["a6f4e21a27bf8d28fc81c873f63f8d78e017666adbf038da0b83c2ad04ef6805",
127         "c02103455f93c2d7ec4b7152db7de00d1c9e806b1945426b6773026b4a85dd03"]],
128         "cc": "d5ac037bb78db41cf924af713b7379c39a4e13901d3eac017238550a1a3b910a"
129       }],
130     "pseudoOuts": [ "b313c1ae9ca06213684fbdefa9412f4966ad192bc0b2f74ed1731381adb7ab58",
131     "7148e7ef5cfd156c62a6e285e5712f8ef123575499ff9a11f838289870522423"]
132   }
133 }
```

## Компоненты транзакции

- (строка 2) - команда `print_tx` указывает блок, в котором была найдена транзакция, что мы и воспроизводим здесь в демонстрационных целях.
- `version` (строка 4) - формат транзакции / версия по времени создания; «2» соответствует протоколу RingCT.
- `unlock_time` (строка 5) - не позволяет тратить выходы транзакции до истечения определённого срока. Это может быть либо высота блока, либо временная метка UNIX, если высота больше, чем начало времени UNIX. По умолчанию устанавливается нулевое значение, но предельного значения не задаётся.
- `vin` (строки 6-23) - список входов (в данном случае их два).
- `amount` (строка 8) - обрезанное (унаследованное) поле суммы, используемое для транзакций первого типа.

- `key_offset` (строка 9) - позволяет верификаторам находить ключи и обязательства участников кольца в блокчейне и убедиться в том, что эти участники являются легитимными. Первый офсет является абсолютным в истории блокчейна, а каждый последующий офсет будет относительным предшествующему. 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. подпункт 6.2.4).

- `k_image` (строка 12) - образ ключа $\tilde{K_j}$ из подпункта 3.5, где $j = 1$, поскольку это первый вход.

- `vout` (строки 24-35) - список выходов (в данном случае их два).

- `amount` (строка 25) - обрезанное поле суммы, используемое для транзакций первого типа.

- `key` (строка 27) - одноразовый ключ адресата для выхода $t = 0$, как было описано в подпункте 4.2

- `extra` (строки 36-39) - дополнительные данные, включающие *в себя публичный ключ транзакции*, то есть общий секрет $rG$ (см. подпункт 4.2) и ID платежа или зашифрованный ID платежа (см. подпункт 4.4. Как правило, это работает следующим образом: каждое число имеет размер один байт (может находиться в диапазоне 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 «1» означает «публичный ключ транзакции». Размер публичных ключей транзакции всегда составляет 32 байта, поэтому нет никакой необходимости указывать их размер. Через 32 числа мы находим новый тег «2», означающий «дополнительный нонс», длина которого равна «9», а следующий байт будет иметь значение «1», что обозначает 8-байтовый зашифрованный идентификатор платежа (в дополнительном нонсе для различных целей может содержаться до пяти полей). Ещё через восемь байтов поле заканчивается. Более подробная информация содержится в работе [81]. (Примечание: в оригинальной спецификации Cryptonote первый байт указывал на размер поля. Monero не использует этого.) [140] src/cryptonote_basic/tx_extra.h

- `rct_signatures` (строки 40-50) - первая часть данных подписи.

- `type` (строка 41) - тип подписи: `RCTTypeBulletproof2` является четвёртым типом. Обрезанными типами RingCT были `RCTTypeFull` и `RCTTypeSimple`, 1 и 2 тип, соответственно. Майнинговые транзакции используют нулевой тип подписи, `RCTTypeNull`.

- `txnFee` (строка 42) - комиссия за проведение транзакции, указываемая простым текстом. В данном случае размер комиссии составляет 0,00003246 XMR.

- `ecdhInfo` (строки 43-47) - «информация по эллиптической кривой Диффи-Хелмана»: скрытая сумма для каждого выхода $t \in \{0, ..., p-1\}$, где $p = 2$.

- `amount` (строка 44) - поле *суммы* для $t = 0$, как было указано в подпункте 5.3

- `outPk` (строки 48-49) - обязательства по каждому выходу (см. подпункт 5.4.
- `rctsig_prunable` (строки 51-132) - вторая часть данных подписи.
- `nbp` (строка 52) - количество доказательств диапазона Bulletproof в данной транзакции.
- `bp` (строки 53-80) - элементы доказательства Bulletproof (доказательства Bulletproofs не рассматриваются в рамках настоящего документа, поэтому мы не вдаёмся в подробности).

$$\Pi_{BP} = (A, S, T_1, T_2, \tau_x, \mu, \mathbb{L}, \mathbb{R}, a, b, t)$$

- `MGs` (строки 81-129) - подписи MLSAG.
- `ss` (строки 82-103) - компоненты $r_{i,1}$ и $r_{i,2}$ подписи MLSAG для первого входа.

$$\sigma_j(\mathfrak{m}) = (c_1, r_{1,1}, r_{1,2}, ..., r_{v+1,1}, r_{v+1,2})$$

- `cc` (строка 104) - компонент $c_1$ вышеупомянутой подписи MLSAG.
- `pseudoOuts` (строки 130-131) - обязательства по псевдовыходам $C'^a_j$, о которых говорится в подпункте 5.3. Пожалуйста, не забывайте о том, что сумма этих обязательств будет равна сумме двух обязательств по выходам данной транзакции (плюс обязательство по комиссии за проведение транзакции $fH$).

# ПРИЛОЖЕНИЕ B

## 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).

```
print_block 1582196
timestamp: 1527458161
previous hash: 30bb9b475a08f2ea6fe07a1fd591ea209a7f633a400b2673b8835a975348b0eb
nonce: 2147489363
is orphan: 0
height: 1582196
depth: 2
hash: 50c8e5e51453c2ab85ef99d817e166540b40ef5fd2ed15ebc863091ca2a04594
difficulty: 51333809600
reward: 4634817937431
{
  "major_version": 7,
  "minor_version": 7,
  "timestamp": 1527458161,
  "prev_id": "30bb9b475a08f2ea6fe07a1fd591ea209a7f633a400b2673b8835a975348b0eb",
  "nonce": 2147489363,
  "miner_tx": {
    "version": 2,
    "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. Но если быть более точными, 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.
- `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, которые приводятся в подпункте 7.2.

- `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) - идентификатор 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` (строки17-40) - транзакция майнера этого блока.

- `version` (строка 18) - формат транзакции / версия по времени создания; «2» соответствует протоколу RingCT.

- `unlock_time` (строка 19) — выход транзакции майнера не может быть потрачен до блока 1 582 256$^{\text{th}}$, пока не будет вычислено ещё 59 блоков (время блокировки составляет 60 блоков, поскольку выход нельзя будет потратить, пока не истечёт время 60 блоков, например, $2 * 60 = 120$ минут).

- `vin` (строки 20-25) — входы транзакции майнера. Здесь нет ни одного, так как транзакция майнера используется для генерирования вознаграждений за вычисление блоков и комиссий за проведение транзакций.

- `gen` (строка 21) — сокращение от generate (генерировать).

- `height` (строка 22) — высота блока, для которой было сгенерировано вознаграждение за вычисление блока для данной транзакции майнера. Вознаграждение за блок может быть сгенерировано только один раз для каждой высоты блока.

- `vout` (строки 26-32) — выходы транзакции майнера.

- `amount` (строка 27) — сумма, распределяемая в транзакции майнера, содержащая вознаграждение за вычисление блока и комиссии за транзакции в этом блоке. Записывается в атомных единицах.

- `key` (строка 29) — одноразовый адрес, указывающий принадлежность суммы, указанной в транзакции майнера.
- `extra` (строки 33-36) — дополнительная информация транзакции майнера, включающая в себя публичный ключ транзакции.
- `type` (строка 38) — тип транзакции (в этом случае это «0» (`RCTTypeNull`), обозначающий транзакцию майнера).
- `tx_hashes` (строки 41-46) — все идентификаторы транзакций, входящих в этот блок (кроме идентификатора транзакции майнера, который будет следующим: `06fb3e1cf889bb972774a8535208d98db164394ef2b14ecfe74814170557e6e9`).

# ПРИЛОЖЕНИЕ C

## Генезис-блок

В этом приложении нами будет представлена структура генезис-блока Monero. Блок содержит 0 транзакций (с ним просто было отправлено первое вознаграждение за вычисление блока пользователю thankful_for_today [127]). 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 [13], или же скрыто пользовались связанным с криптовалютой программным обеспечением или сервисами [134].

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 [127].

```
print_block 0
timestamp: 0
previous hash: 0000000000000000000000000000000000000000000000000000000000000000
nonce: 10000
is orphan: 0
height: 0
depth: 1580975
hash: 418015bb9ae982a1975da7d79277c2705727a56894ba0fb246adaabb1f4632e3
difficulty: 1
reward: 17592186044415
{
```

```
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 B, 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 идентификатора 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), о чём говорится в подпункте 7.3.1.
- `key` (строка 29) — самые первые Monero были переданы основателю thankful_for_today.
- `extra` (строки 33-36) — благодаря шифрованию, описанному в Приложении A, поле `extra` транзакции майнера 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 39.