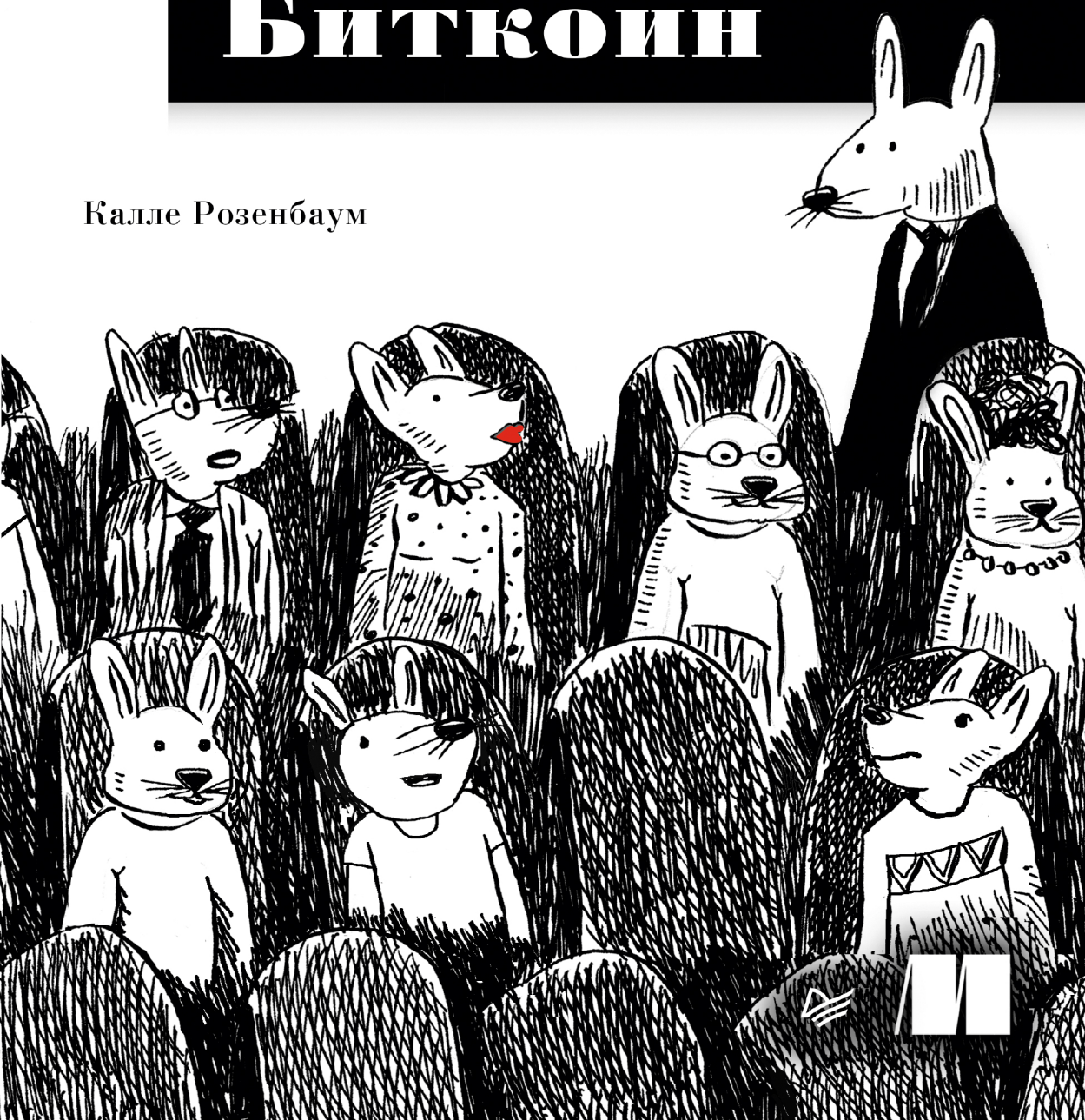
грокаем

технологию Биткоин

Калле Розенбаум



grokking Bitcoin

Kalle Rosenbaum

MANNING
SHELTER ISLAND

грокаем технологию биткоин

Калле Розенбаум

ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск

2020

Калле Розенбаум

Грокаем технологию Биткоин

Серия «IT для бизнеса»

Перевел на русский *А. Киселев*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*К. Тульцева*
Литературный редактор	*А. Руденко*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, Г. Шкатова*
Верстка	*Л. Егорова*

ББК 32.988.02-018.2
УДК 004.77

Розенбаум Калле

Р64 Грокаем технологию Биткоин. — СПб.: Питер, 2020. — 496 с.: ил. — (Серия «IT для бизнеса»).

ISBN 978-5-4461-1424-5

Хотите разобраться в технологии Биткоин на глубоком концептуальном уровне?

«Грокаем технологию Биткоин» на наглядных схемах и ярких примерах учит мыслить по-новому. Вы узнаете, как на самом деле происходит майнинг, возникают биткоины, как войти в сеть Биткоин и как функционирует цифровой кошелек.

Основы технологии Биткоин; хеш-функции и цифровые подписи; криптография и алгоритмы шифрования; анатомия транзакций; верификация и пропускная способность; хард- и софт-форки.

От вас не требуется продвинутых навыков программирования, но базовое представление об основах — базах данных, компьютерных сетях, веб-серверах и (о ужас!) математики — не будет лишним.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1617294648 англ.
ISBN 978-5-4461-1424-5

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.
Дата изготовления: 03.2020. Наименование: книжная продукция. Срок годности: не ограничен.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.
Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.
Подписано в печать 21.02.20. Формат 70×100/16. Бумага офсетная. Усл. п. л. 39,990. Заказ 0000.

Оглавление

Посвящается любви всей моей жизни —
жене Линнее.
Умной, верной, настоящей.
И биткоинерам всего мира

Предисловие

В чем разница между системой Биткоин и аналогами денег, которые используются в многопользовательских играх? Все они определяются программно, создаются из ничего и могут иметь ценность, сопоставимую с традиционными валютами. Случайному наблюдателю они кажутся почти одинаковыми.

Но есть важные различия в том, как они реализованы, что оказывает влияние на все аспекты соответствующих систем: если игровые валюты реактивно банят пользователей, уличенных в мошенничестве, то Биткоин действует упреждающе, предотвращая различные виды мошенничества, делая их невозможными, непрактичными или менее выгодными, чем честное поведение.

Книга «Грокаем технологию Биткоин» точно описывает, как это достигается. Когда вы познакомитесь с хеш-функциями, цифровыми подписями, доказательствами работы и многими другими понятиями, то увидите, что технология Биткоин весьма эффективно предотвращает мошенничество, не требуя глобального запрета на активность какого-либо пользователя.

Здесь приведено множество преимуществ этой ключевой особенности. Когда вы узнаете об адресах, кошельках, майнерах и узлах, то увидите, что в ситуации, когда никто не имеет реальной возможности обманывать, можно разрешить всем участвовать в каждой части протокола, не ставя никого в положение безграничного доверия. И поскольку никто не сможет воспользоваться своим положением, отпадает необходимость в идентификации личности. «Грокаем технологию Биткоин» наглядно демонстрирует, как любой может использовать публичную цепочку блоков (блокчейн), сохраняя определенную степень конфиденциальности.

Децентрализованная система, которая не использует идентификацию личности или не зависит от доверия других участникам, в корне отличается

от привычных систем, знакомых большинству из нас. Калле Розенбаум активно использует эту особенность, начиная объяснение каждого аспекта Биткоина с примера централизованной системы, эксплуатирующей доверие или зависящей от идентификации и понятной любому читателю. Затем он постепенно превращает эту базовую систему в децентрализованную и псевдонимную систему, не требующую выражения доверия и реализованную с использованием технологии Биткоин. Благодаря пошаговым объяснениям с четкими определениями и превосходными иллюстрациями, даже самые узкоспециальные темы становятся доступными для заинтересованного непрофессионала.

Технология Биткоин нуждается в книгах, таких как «Грокаем технологию Биткоин», но ей также нужны активные пользователи, которые читают эти книги и разбираются в технических принципах, на которых она основана. В течение первых дней, которые, как я надеюсь, продолжатся долгой историей использования биткоинов, пользователей часто просят оценить предлагаемые изменения в системе — изменения, которые могут повлиять на безопасность и конфиденциальность их электронных кошельков. Прочитав эту книгу, вы поймете, как система предотвращает мошенничество, и сможете гарантировать, что будущие изменения сохранят эту важную особенность и ее многочисленные преимущества.

Дэвид А. Хардинг (David A. Harding),
соавтор документации для Биткоин

Вступление

Биткоин изменил мир.

Мир стал свидетелем крупных достижений, способствовавших улучшению качества жизни людей, в том числе изобретения вакцины, электричества, радио, автомобиля и Интернета. Некоторые из этих технологий начинались как игры для богатых, но в конечном счете получили массовое распространение и стали приносить огромную пользу жителям Земли. Биткоин вскоре встанет с ними в один ряд. Именно это и кажется мне в нем самым захватывающим.

Мне повезло жить в социально стабильном обществе. Проводя финансовые операции, я не боюсь, что кто-то постучится в мою дверь. Я никогда не чувствовал, что нужно как можно быстрее потратить деньги, потому что гиперинфляция съест мой обед. Но из-за этого мне трудно было понять важность Биткоин. Технология Биткоин[1] интересует меня больше с теоретической стороны, но когда я слышу от менее удачливых людей, живущих под властью репрессивных или некомпетентных режимов, о том, как Биткоин улучшает их жизнь, все становится более чем реальным. Биткоин даст людям возможность отказаться от системы, которая держит их в заложниках.

Сатоши Накамото (Satoshi Nakamoto), так называет себя человек (или группа людей), пожелавший не раскрывать свою личность, опубликовавший в октябре 2008 года научную статью в списке рассылки, посвященном

[1] Здесь и далее: *Биткоин* (с прописной буквы): концепция и протокол первой децентрализованной криптовалюты, использующей технологию блокчейн. Криптовалютный токен *биткоин* (со строчной): валюта сама по себе. — *Примеч. ред.*

криптографии. Статья называлась «Bitcoin: A Peer-to-Peer Electronic Cash System» (см. приложение B). В своей статье Накамото описывает важные части технологии Биткоин, первой системы электронных (цифровых) денег, в которой отсутствует центральный орган эмиссии или обработки транзакций. В январе 2009 года Накамото опубликовал первую программу, реализующую эту систему. В то время технология Биткоин не привлекла широкого внимания, кроме узкого круга экспертов в области криптографии. Но по мере того как система доказывала свои возможности, круг интересующихся расширялся. Однако пока голос противников Биткоин звучит громче, чем голос сторонников, как часто бывает с новаторскими технологиями. По состоянию на 2019 год уже сотни миллионов людей знают о Биткоин и десятки миллионов используют эту систему.

Когда я начал изучать Биткоин в 2013 году, мне потребовалось много времени, чтобы получить достаточно полное понимание технологии. И не потому, что я несообразительный, а потому что Биткоин — сложная система. Это не просто новомодная база данных, а смесь экономики, математики, технологий и антропологии.

В 2015 году я начал вести блог о Биткоин, и, думаю, издательству Manning понравились мои статьи, потому что компания прислала мне электронное письмо с предложением написать книгу о «блокчейне». Поскольку моя страсть — Биткоин, а не только блокчейн, чрезмерно раскрученное слово, обозначающее базу данных для Биткоин, — я выразил свою благодарность и сказал, что мне было бы интереснее написать книгу о Биткоин. Я несколько лет искал свое место в биткоин-сообществе, и эта возможность показалась мне подходящей. Я начал работу над проектом, но он оказался намного сложнее и занял больше времени, чем я ожидал.

Первоначально эта книга задумывалась как типичный технический обзор Биткоин, но оказалось трудно описать одну тему без передачи всех нюансов. Начало книги получилось бы слишком сложным. Стало ясно, что нужен какой-то иной подход. Я обсудил это с женой, и у нас появилась идея концептуального описания Биткоин с нуля. Книга начинается с действительно простой системы электронных таблиц, понятной каждому, и эта система постепенно превращается в Биткоин. В каждой главе описываются некоторые проблемы в текущей системе, и затем решаются добавлением технологии, рассматриваемой в этой главе.

открытой лицензией был моим требованием, которое я выдвинул, перед тем как взяться за эту книгу. Это мой способ внести свой вклад в биткоин-сообщество, давшее мне так много за эти годы. Я и издательство Manning надеемся, что те, кто не сможет позволить себе купить эту книгу, смогут прочитать ее свободно распространяемую версию. Однако благотворительность — не единственная причина. Мы также надеемся, что эта версия сделает книгу более заметной, дав потенциальным читателям возможность ознакомиться с ее содержимым. Если можете себе это позволить, пожалуйста, купите красивую печатную версию. Издательство Manning и я потратили много времени и сил на создание этой книги, и мы будем очень признательны за ваш вклад!

Я надеюсь, вам понравится книга «Грокаем технологию Биткоин» настолько же, насколько я страдал, работая над ней!

Благодарности

Я должен сказать спасибо многим людям, без которых эта книга была бы невозможна.

В первую очередь хочу поблагодарить свою жену Линнею за наши обсуждения и ее безоговорочную поддержку. Ты лучшая!

Кристина Тейлор (Christina Taylor), мой дорогой редактор из Manning, спасибо, что не давали мне расслабиться. Работа, работа и еще раз работа. Спасибо также вам, Берт Бейтс (Bert Bates), за ваш ценный вклад в педагогику.

Я также хочу поблагодарить других сотрудников издательства Manning за их профессионализм: Озрена Харловича (Ozren Harlovic), Ребекку Райнхарт (Rebecca Rinehart), Кэндис Гиллхулли (Candace Gillhoolley), Ану Ромак (Ana Romac), Майкла Стивенса (Michael Stephens), Эрин Туи (Erin Twohey), Кристофера Кауфмана (Christopher Kaufmann), Матко Хрватина (Matko Hrvatin) и Грега Уайлда (Greg Wild). Отдельное спасибо фантастической команде из технического отдела: Чаку Ларсону (Chuck Larson), иллюстратору; Ребекке Деуэль-Гальегос (Rebecca Deuel-Gallegos), редактору; Кевину Каллахану (Kevin Callahan), наборщику; Тиффани Тейлор (Tiffany Taylor), корректору; а также Лори Вайдерт (Lori Weidert), выпускающему редактору.

Хочу поблагодарить Джонатана Йогенфорса (Jonathan Jogenfors) и Понта Линдблома (Pontus Lindblom) за обзор некоторых глав, охватывающих их области знаний.

Многие люди на различных интернет-форумах оказали значительную помощь мне в моих исследованиях. Особенно помогли Дэвид А. Хардинг (David A. Harding), Питер Уилль (Pieter Wuille) и Марк Эрхардт (Mark

«Murch» Erhardt). Спасибо вам за беседы в Twitter и отличные ответы на Bitcoin Stack Exchange (см. веб-ресурс 2 в приложении C).

В обсуждении условий открытой публикации этой книги мне помогал Андреас М. Антонопулос (Andreas M. Antonopoulos). Вашу поддержку в этом деле трудно переоценить. Спасибо! Также спасибо за аналогию с «вегетарианским рестораном», использованную в главе 11. И спасибо за увлекательные беседы с вами; вы стали для меня главным источником вдохновения.

Я писал эту книгу, используя Asciidoctor — язык разметки текста. Спасибо вам, Дэн Аллен (Dan Allen), за ваш тяжелый труд над Asciidoctor; это невероятно удобный язык.

Книга преодолела три раунда рецензирования с разными научными редакторами. Их отзывы и замечания были чрезвычайно полезны для проверки идей и выявления пробелов. Спасибо вам: Ян Гойвертс (Jan Goyvaerts), Макс Хамбер (Max Humber), Ирина Романенко (Iryna Romanenko), Жан-Франсуа Морин (Jean-François Morin), Аль Кринкер (Al Krinker), Джоэл Котарски (Joel Kotarski), Маркус Бекманн (Markus Beckmann), Кристофер Бейли (Christopher Bailey), Витон Витанис (Viton Vitanis), Паоло Фрейли (Paolo Freuli), Томо Хельман (Tomo Helman), Марчелло Сери (Marcello Seri), Мацей Дроздзовски (Maciej Drozdzowski), Цицерон Зандона (Cicero Zandona), Барнаби Норман (Barnaby Norman), Фрэнсис Буонтемпо (Frances Buontempo), Гленн Свонк (Glenn Swonk) и Серхио Фернандес Гонсалес (Sergio Fernandez Gonzalez).

Спасибо также всем читателям программы раннего доступа Manning Early Access Program (MEAP), которые делились своими мыслями, предложениями и вопросами на форуме издательства Manning, а также спасибо Аруну Сурье (Aruna Surya) за советы и предложения, присланные по электронной почте.

Если я кого-то не упомянул, прошу извинить меня — и вам тоже спасибо.

О книге

Основная цель этой книги — помочь вам решить, доверяете ли вы технологии Биткоин. На пути к этой цели вы подробно познакомитесь с множеством понятий, таких как цифровые подписи, доказательство работы и одноранговые сети. Попутно мы достигнем некоторых промежуточных целей:

- установим кошелек с биткоинами на свой смартфон и разберемся, как все это работает;
- изучим технические аспекты Биткоин;
- рассмотрим варианты хранения личных ключей, в зависимости от количества биткоинов в кошельке и желаемого уровня безопасности и удобства;
- запустим полноценный узел для участия в финансовых транзакциях без третьих сторон;
- познакомимся с необычными заявлениями жуликов, обманщиков и аферистов, сопровождающими успехи Биткоин. Будьте внимательны!

Кому адресована эта книга

Эта книга адресована всем, кто хочет понять технологию Биткоин на глубоком концептуальном уровне. Книга не требует навыков программирования, но базовое представление о некоторых технических понятиях пригодится — например, о базах данных, компьютерных сетях, компьютерных программах и веб-серверах. Нелишними будут и некоторые знания математики, но это, конечно, не обязательно.

Структура книги

Эта книга состоит из 11 глав и трех приложений.

- Глава 1 предлагает обзор Биткоин. Здесь вы узнаете, что такое технология Биткоин, почему она так важна и как работает.
- Глава 2 рассматривает криптографические аспекты: хеш-функции и цифровые подписи. Это основные строительные блоки, знание которых понадобится в остальной части книги. Здесь также закладывается основа системы фиктивных денег на примере таблицы жетонов на булочки, которую мы будем развивать в главах 2–8.
- Глава 3 охватывает адреса. Пересылая биткоины, вы должны указать биткоин-адрес получателя. Что такое биткоин-адреса, зачем они нужны, как создаются и используются?
- Глава 4 рассказывает, как биткоин-кошелек следит за вашими секретными ключами и как сгенерировать несколько секретных ключей из одного огромного случайного числа, называемого начальным числом, или сидом. Здесь также подробно обсуждаются вопросы резервного копирования.
- Глава 5 исследует анатомию транзакций в Биткоине, как транзакции получают цифровую подпись и обрабатываются.
- Глава 6 рассматривает технологию блокчейн (blockchain): базу данных для хранения транзакций. Здесь мы посмотрим, как устроена цепочка блоков и как она позволяет использовать так называемые легкие кошельки.
- Глава 7 охватывает вопросы доказательства работы, которое используется для выбора того, кто добавит новую транзакцию в цепочку блоков. Этот процесс, называемый майнингом, обеспечивает безопасность биткоинов в цепочке блоков — блокчейне.
- Глава 8 исследует сеть Биткоин. Биткоин не имеет центрального органа управления, и вы узнаете, как можно организовать подобное с использованием одноранговой сети. Здесь также рассказывается, как стать активным членом сети Биткоин, запустив свой узел.
- Глава 9 вновь возвращается к транзакциям, чтобы рассказать о дополнительных возможностях, важных для разных применений.
- Глава 10 вводит понятие протокола SegWit. В 2017-м в технологию Биткоин было добавлено серьезное усовершенствование, повышающее надежность транзакций, эффективность верификации и пропускную способность блокчейна, и в этой главе мы узнаем все подробности.

- Глава 11 рассматривает ситуации софт-форка и хард-форка, то есть разделения версий криптовалюты, и того, как Биткоин может безопасно модернизироваться путем софт-форка с использованием тщательно разработанного плана развертывания.

Советую обязательно прочитать главы 2–8, в которых мы последовательно с самого начала будем строить систему жетонов, или токенов, на булочки. Каждая глава добавляет в систему технологию для решения конкретной проблемы, и к концу главы 8 мы создадим сеть Биткоин. Главы 9, 10 и 11 можно читать не по порядку, но рекомендую особенно внимательно прочитать главу 11, потому что она охватывает суть Биткоин. Поняв, о чем рассказывает глава 11, вы грокнете Биткоин.

Время от времени я буду использовать некоторые обзорные рисунки из главы 1, как во вступительных разделах к главам, так и внутри самих глав, чтобы вы не заблудились. Легко потерять представление об общей картине и цели текущей темы; ищите заголовки со значком перископа «На чем мы остановились?»

Все главы, кроме главы 1, включают упражнения. Я добавил их, чтобы вы могли оценить свои знания и навыки. Каждый блок упражнений делится на разделы с простыми, помогающими быстро проверить знание некоторых фактов, и более сложными, требующими более скрупулезных рассуждений, упражнениями. Называются они «Для разминки» и «Придется пораскинуть мозгами». Некоторые упражнения в разделах «Придется пораскинуть мозгами» действительно очень сложные, поэтому, если вам не удастся их выполнить, обратитесь к приложению B за ответами.

Соглашения об оформлении кода

В книге очень мало примеров программного кода. Его практически нет. Но в главе 8 и в приложении A приводятся несколько команд для ОС Linux. Команды начинаются со знака доллара и пробела, как показано ниже:

```
$ cd ~/.bitcoin
```

Длинные команды, не умещающиеся в одну строку, будут разбиты на несколько строк с использованием символа обратного слеша \ и отступа в следующей строке на четыре пробела, например:

```
$ ./bitcoin-cli getrawtransaction \
    30bca6feaf58b811c1c36a65c287f4bd393770c23a4cc63c0be00f28f62ef170 1
```

Обратные слеши используются для записи команд в нескольких строках большинством интерпретаторов командной строки для Linux, поэтому команды из книги можно напрямую копировать в окно терминала. Длинные строки, которые выводятся командами, не разбиваются с использованием обратного слеша; они просто переносятся и снабжаются значком стрелки, обозначающим перенос, как показано ниже:

```
{"result":"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
↪"error":null,"id":"1"}
```

Во всей книге данные оформляются моноширинным шрифтом, например: `7af24c99`. Я обычно не указываю кодировку явно (десятичные числа, шестнадцатеричные строки, строки base64, строки base58 и т. д.), потому что часто это очевидно из контекста.

Прочие онлайн-ресурсы

Если у вас есть вопросы о Биткоин, на которые вы не нашли здесь ответов, рекомендую обратиться на сайт Bitcoin Stack Exchange (приложение B), где качество ответов определяется голосованием читателей.

Также советую посетить сайт Bitcoin Developer Reference (веб-ресурс 3), где можно найти исчерпывающую документацию о Биткоин.

Исходный код Bitcoin Core (веб-ресурс 4) является наиболее точным источником информации. Это эталонная реализация протокола Биткоин, и чтение такого исходного кода иногда является единственным способом найти необходимые ответы.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Об авторе

Калле Розенбаум (Kalle Rosenbaum) занимается разработкой программного обеспечения вот уже 20 лет. Его увлечение Биткоином началось в 2013 году и продолжается до сих пор. В 2015-м Калле основал консалтинговую компанию по технологии Биткоин и с тех пор работает в этой отрасли. Он также ведет блог, где разъясняет различные технические вопросы, связанные с Биткоином, такие как усовершенствование механизма распространения блоков, боковые цепочки и транзакции «replace-by-fee» (замена за плату). Его блог служит целью помочь себе и другим извлекать выгоду.

1 Введение в Биткоин

Эта глава охватывает следующие темы:

- ✓ знакомство с технологией Биткоин;
- ✓ оплата биткоинами;
- ✓ проблемы, решаемые технологией Биткоин.

Цель этой книги — дать достаточный объем сведений о технологии Биткоин, чтобы вы смогли принять обоснованное решение о том, как можно использовать ее для улучшения личной жизни или бизнеса. Надеюсь, вы узнаете достаточно, чтобы решить, доверяете ли вы Биткоин или нет (надеюсь, что первое). Начиная движение к этой цели, я буду предполагать, что вы знаете, что означают следующие термины:

- ✶ компьютерная программа;
- ✶ база данных;
- ✶ компьютерная сеть;
- ✶ веб-сервер.

Если какой-то из этих терминов вам не знаком, не волнуйтесь. Отыщите его определение либо продолжайте чтение — я думаю, что вы не будете испытывать особых затруднений.

Что такое Биткоин?

Биткоин — это система электронных (цифровых) денег, которая позволяет людям обмениваться биткоинами (денежная единица в системе Биткоин)

между собой, не прибегая к услугам банков или какой-либо другой доверенной третьей стороны. Биткоины похожи на обычные банкноты и монеты, но это исключительно электронная валюта, и операции с ней осуществляются через Интернет. Валюта Биткоин не привязана ни к какой *фиатной валюте*, такой как доллар США или китайский юань; она имеет свободный плавающий обменный курс по отношению к большинству фиатных валют. Вы можете покупать и продавать биткоины за бумажные валюты онлайн, используя одну из нескольких бирж, например kraken.com, bitstamp.net или localbitcoins.com.

КАК ПРАВИЛЬНО?

Система называется *Биткоин* (*Bitcoin*), и ее название записывается с заглавной буквы *Б* (*B*). Денежная единица называется *биткоин* (*bitcoin*) и записывается с маленькой буквы *б* (*b*). Часто для обозначения биткоинов используется значок ₿ и аббревиатуры BTC и XBT. В этой книге я буду использовать аббревиатуру BTC.

Никакие компании или правительства не контролируют систему Биткоин. Ее работу круглосуточно обеспечивают тысячи компьютеров по всему миру, объединенных в *сеть Биткоин*, как показано на рис. 1.1. Вам не нужно нигде регистрироваться — чтобы использовать Биткоин, достаточно иметь доступ в Интернет и компьютерную программу, например мобильное приложение.

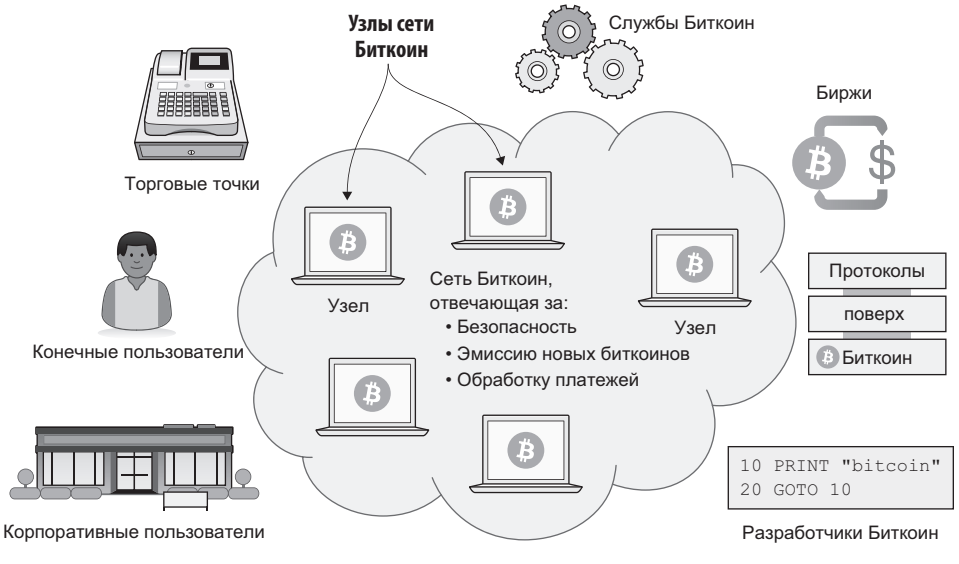


Рис. 1.1. Сеть Биткоин и ее экосистема

Любой желающий может использовать или участвовать в сети Биткоин без специального разрешения со стороны банка или другого подобного учреждения. Благодаря *свободной* природе Биткоин, за последние годы появилось много родственных технологий. Участников этой биткоин-экосистемы можно грубо разделить на несколько групп:

КАКИЕ АСПЕКТЫ НЕ ИНТЕРЕСУЮТ БИТКОИН

Сеть Биткоин не проводит различий между пользователями. Никто не находится в более привилегированном положении, чем другие. Неважно, кто они и чем занимаются, все участвуют в работе сети на общих условиях.

- *Конечные пользователи* — люди, использующие Биткоин для своих повседневных нужд, например для хранения сбережений, покупок, игры на бирже или заработка.
- *Корпоративные пользователи* — компании, использующие Биткоин для решения своих бизнес-задач, например для выплаты заработной платы работникам в других странах или в случаях, аналогичных конечным пользователям.
- *Торговые точки* — например, рестораны или книжные магазины, принимающие платежи в биткоинах.
- *Службы Биткоин* — компании, предоставляющие клиентам услуги, связанные с Биткоин, такие как пополнение счета мобильного телефона, услуги анонимного доступа, денежные переводы или выплата чаевых.
- *Биржи* — коммерческие службы, которые люди могут использовать для обмена местной валюты на биткоины и обратно.
- *Протоколы поверх Биткоин* — системы, действующие «поверх» Биткоин и выполняющие определенные задачи, например протоколы платежных сетей, специализированные маркеры и децентрализованные биржи.
- *Разработчики Биткоин* — люди, работающие (часто бесплатно) над компьютерными программами с открытым исходным кодом, которыми пользуются участники сети Биткоин.

Задача сети Биткоин заключается в обработке платежей в биткоинах, ведении учета того, кто чем владеет, защите от несанкционированных модификаций и эмиссии новых биткоинов с заранее установленной скоростью. Сеть состоит из тысяч компьютеров, разбросанных по всему миру. Мы называем эти компьютеры *узлами Биткоин*, или просто *узлами*. Любой из участников, упомянутых выше, тоже может активно участвовать в работе биткоин-сети, запустив собственный биткоин-узел. Вы тоже должны будете запустить свой узел, если не хотите доверять другим предоставление вам финансовой информации.

Общая картина

Сеть Биткоин — это сеть из компьютеров, на которых установлено программное обеспечение Биткоин. Эта сеть проверяет и подтверждает платежи между пользователями Биткоин.

Предположим, что Алиса хочет заплатить Бобу 1 BTC. Платеж начинается с того, что Алиса создает транзакцию и отправляет ее в сеть Биткоин, как показано на рис. 1.2. Здесь я изобразил в общих чертах четыре этапа и каждый из них объясню в следующих разделах. Рисунок 1.2 появится также во введении к главам 2–8, где я также укажу, какую часть рисунка мы будем рассматривать в той или иной главе.

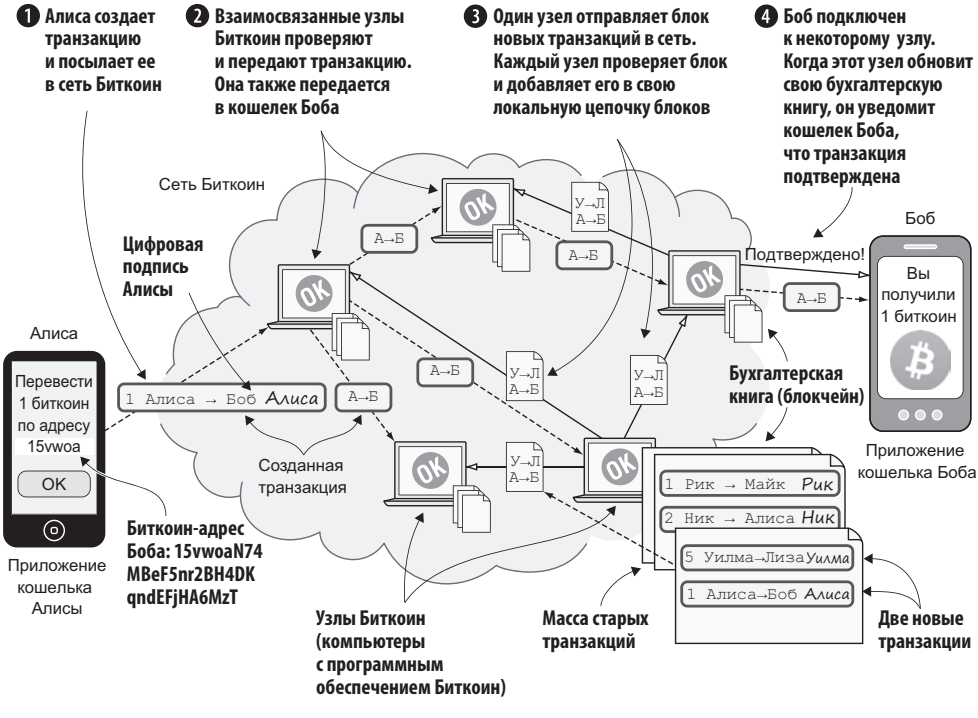


Рис. 1.2. Выполнение платежа в сети Биткоин. Платеж обрабатывается в четыре этапа

Теперь посмотрим, как обрабатывается платеж на пути от Алисы к Бобу:

❶ Алиса создает транзакцию, которая переводит 1 биткоин Бобу, подписывает ее и посылает в сеть Биткоин.

❷ Компьютеры в сети — узлы Биткоин — проверяют, действительно ли у Алисы есть деньги и является ли транзакция подлинной. Затем они передают транзакцию своим соседям, которые называют *пирами*.

❸ Каждый компьютер обновляет свою копию *цепочки блоков Биткоин* (*блокчейн*), или *консенсусный реестр*, добавляя в нее информацию о новом платеже.

❹ Сеть уведомляет Боба, что он получил 1 биткоин.

Я ДУМАЛ, ЧТО БИТКОИН ПОДДЕРЖИВАЕТ АНОНИМНОСТЬ!

Биткоин не использует имен или другую личную информацию. Я указал здесь имена участников только ради упрощения.

Обратите внимание, что в действительности Алиса *не посылает* 1 биткоин Бобу, а просит сеть переместить 1 биткоин из кошелька Алисы в кошелек Бобу внутри цепочки блоков (блокчейне) Биткоин.

Цепочка блоков (блокчейн) Биткоин — это база данных, копия которой хранится на каждом компьютере в сети Биткоин. Представьте, что блокчейн — это большая бухгалтерская книга, в котором зафиксированы все транзакции, выполнявшиеся когда-либо.

В следующих четырех разделах мы подробно разберем каждый этап.

Шаг ❶: транзакции

Шаг ❶ (рис. 1.3) — Алиса просит сеть перевести 1 биткоин Бобу. Для этого она посылает транзакцию в сеть Биткоин. Транзакция содержит инструкции о том, как перевести деньги, и цифровую подпись, подтверждающую, что именно Алиса просит перевести деньги.

ТРАНЗАКЦИЯ

Транзакция — это платеж. Эти два термина взаимозаменяемы. Подробнее о транзакциях рассказывается в главах 5 и 9.

Транзакция Биткоин — это блок данных, определяющий:

- сумму перевода (1 биткоин);
- биткоин-адрес получателя перевода (биткоин-адрес Боба, `15vwoaN74MBeF5nr2BH4DKqndEFjHA6MzT`);
- *цифровую подпись* (созданную с помощью закрытого ключа Алисы).

Рис. 1.3. Алиса создает транзакцию, подписывает ее и посылает одному или нескольким узлам в сети Биткоин

Цифровая подпись создается на основе транзакции и огромного секретного числа, называемого *закрытым ключом*, известного только Алисе. В результате получается цифровая подпись, которую может создать только владелец закрытого ключа.

Приложение мобильного кошелька Алисы подключается к одному или нескольким узлам в сети Биткоин и посылает им созданную транзакцию.

ЦИФРОВЫЕ ПОДПИСИ

Подробнее о цифровых подписях рассказывается в главе 2.

Шаг ❷: сеть Биткоин

Алиса посылает транзакцию одному или нескольким узлам Биткоин. На этапе ❷ (рис. 1.4) каждый такой узел проверяет действительность транзакции и передает ее своим пирам. Для этого он обращается к локальной копии блокчейна (цепочки блоков) и убеждается, что:

- биткоин, который пытается потратить Алиса, действительно существует;
- цифровая подпись Алисы действительна.

Если все проверки пройдены, узел пересылает транзакцию своим пирам в сети Биткоин. Такая пересылка называется *ретрансляцией*. Спустя какое-то время транзакция Алисы пройдет через всю сеть и каждый узел проверит ее по пути. Блокчейн пока не обновляется; это следующий шаг.

ОШИБОЧНЫЕ ТРАНЗАКЦИИ

Ошибочные транзакции просто уничтожаются. Они не уходят дальше первого узла.

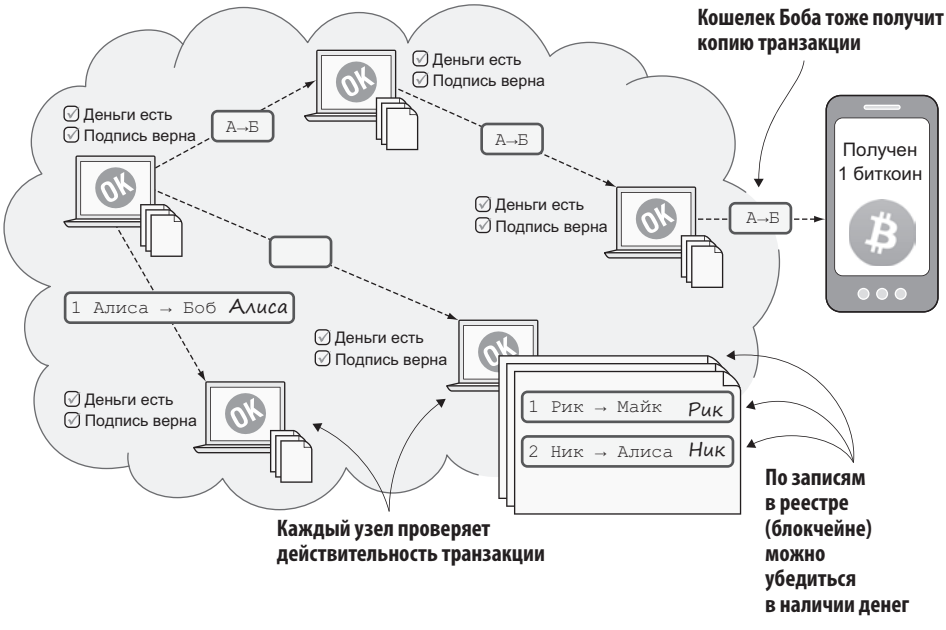


Рис. 1.4. Алиса посылает транзакцию узлу в сети. Узел проверяет транзакцию и пересылает ее другим узлам. В какой-то момент транзакция будет получена всеми узлами сети

Шаг ❸: блокчейн

На шаге ❸ узлы обновляют свои локальные копии цепочки блоков (блокчейна) Биткоин, добавляя транзакцию Алисы. Блокчейн содержит историческую информацию обо всех предыдущих транзакциях; время от времени в него добавляются новые транзакции, такие как транзакция Алисы.

БЛОКЧЕЙН

Название *блокчейн* (*цепочка блоков*) описывает структуру консенсусного реестра. Консенсусный реестр организован как последовательность блоков, составленных в цепочку так, чтобы легко было обнаружить любые изменения. Подробнее об этом рассказывается в главе 6.

Добавление транзакции Алисы в блокчейн выполняется не так просто, как может показаться. Транзакция Алисы не единственная в сети Биткоин.

Одновременно могут выполняться тысячи транзакций. Если все узлы будут обновлять свои копии блокчейна при получении каждой транзакции, копии недолго будут оставаться копиями, потому что на разные узлы транзакции могут приходить в разном порядке, как показано на рис. 1.5.

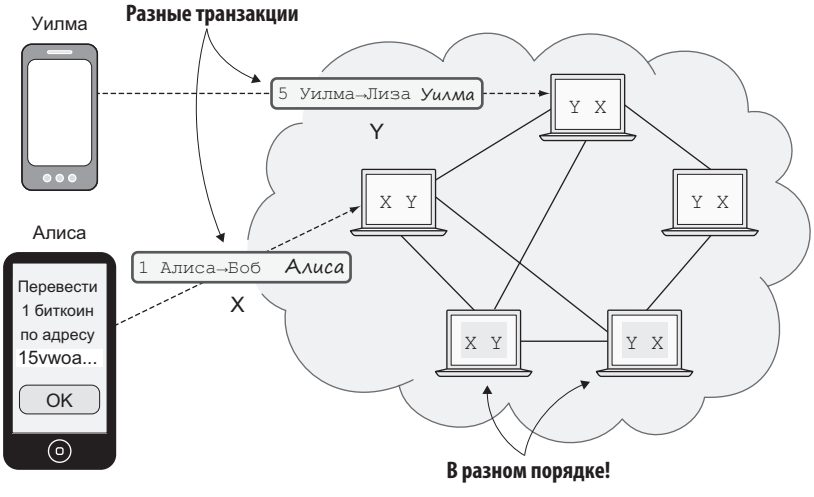


Рис. 1.5. На разные узлы транзакции поступают в разном порядке. Если все узлы будут записывать транзакции в блокчейн в порядке их поступления, блокчейны на разных узлах будут отличаться

Для упорядочения транзакций один узел берет на себя инициативу, говоря: «Я хочу добавить эти две транзакции в блокчейн в порядке Y, X!» Это сообщение, известное как *блок*, отправляется лидером (рис. 1.6) точно так же, как Алиса отправляет транзакцию.

Получив новый блок, узлы обновляют свои копии блокчейна и передают блок своим пирам. Транзакция Алисы — одна из транзакций в блоке и теперь становится частью блокчейна.

ДОБАВЛЯТЬ НОВЫЕ ТРАНЗАКЦИИ МОЖНО ТОЛЬКО В КОНЕЦ БЛОКЧЕЙНА

Новые транзакции всегда добавляются в конец блокчейна — он растет только с конца.

Почему узел может пожелать взять на себя инициативу? Узел, берущий инициативу, вознаграждается вновь отчеканенными биткоинами и комиссией за транзакции, которые он включает в блок.

Сможет ли любой узел все время брать на себя инициативу и получать вознаграждение? Нет, не сможет. Чтобы взять на себя инициативу, узел должен

решить сложную задачу. Для этого требуется много времени и электроэнергии, а значит, лидеры будут возникать нечасто. Задача настолько сложна, что большинство узлов в сети даже не пытаются браться за нее. Узлы, делающие такую попытку, называют *майнерами*[1], потому что они добывают новые монеты, подобно тому как золотоискатель добывает золото. Более подробно мы обсудим этот процесс в главе 7.

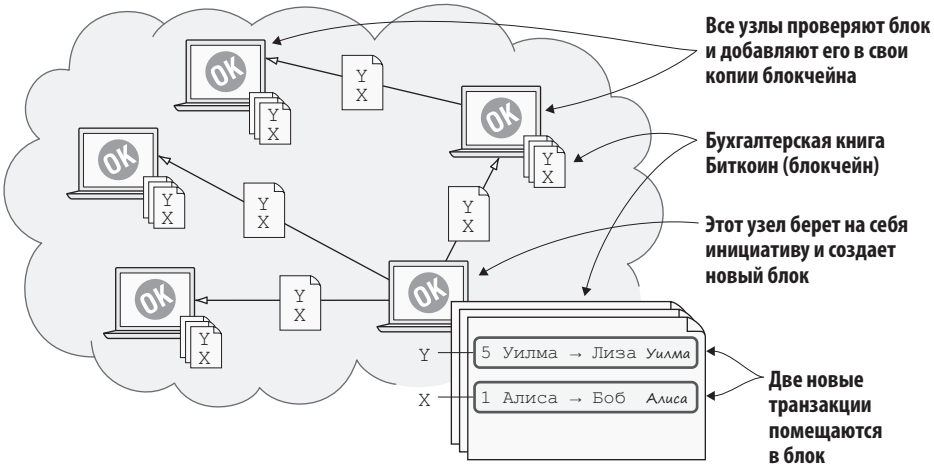


Рис. 1.6. Один из узлов берет на себя инициативу и определяет порядок добавления транзакций. Другие узлы проверяют блок и обновляют свои копии блокчейна

Шаг ❹: кошельки

Боб и Алиса являются пользователями сети Биткоин, и им обоим нужна компьютерная программа для взаимодействия с сетью. Такая программа называется *биткоин-кошелек*. Существует несколько типов биткоин-кошельков для разных устройств, таких как мобильные телефоны и настольные компьютеры, и даже существуют специализированные аппаратные кошельки.

ЧТО ДЕЛАЕТ КОШЕЛЕК

Типичный биткоин-кошелек:

- управляет ключами;
- следит за входящими/исходящими биткоинами;
- посылает биткоины.

[1] В переводе с англ. *miner* (майнер) — добытчик. — *Примеч. пер.*

Перед шагом ❹ процесса оплаты узлы в сети обновляют свои локальные копии блокчейна. Теперь сеть должна уведомить Алису и Боба о том, что транзакция прошла, как показано на рис. 1.7.

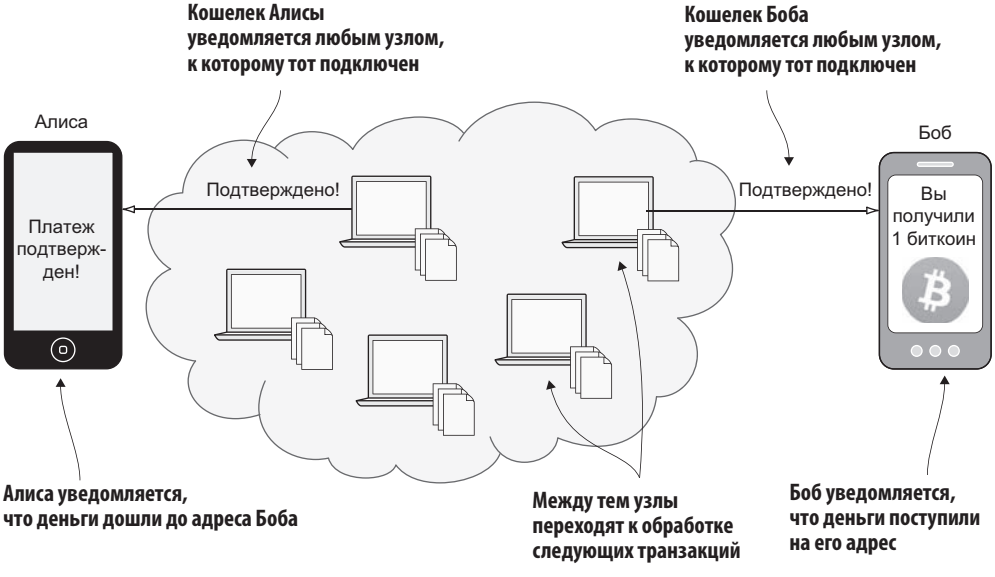


Рис. 1.7. Кошелек Боба просит узел сообщить об операциях с биткоин-адресом Боба. Алиса послала платеж по адресу Боба, а узел только что записал транзакцию в блокчейн, о чем уведомляет кошелек Боба

Кошелек Боба подключен к некоторым узлам сети Биткоин. Когда в блокчейн добавляется транзакция, касающаяся Боба, узлы, к которым подключен кошелек Боба, уведомят его об этом. Кошелек Боба отобразит сообщение, что он получил 1 биткоин. Алиса тоже использует кошелек. Ее кошелек также будет уведомлен о выполнении транзакции.

Помимо отправки и получения транзакций, кошельки Алисы и Боба также управляют их закрытыми ключами. Как отмечалось выше, закрытый ключ используется для создания цифровых подписей биткоин-адреса. Алиса создала свою цифровую подпись с помощью одного из своих закрытых ключей. Когда позже Боб захочет потратить деньги, которые получил на свой биткоин-адрес, сгенерированный с использованием его закрытого ключа, он должен будет создать транзакцию и подписать ее цифровой подписью этим закрытым ключом.

Проблемы современных денег

Технология Биткоин не получила бы такого распространения, если бы не решала насущных проблем. Биткоин решает некоторые проблемы, свойственные традиционным финансовым системам. Давайте рассмотрим некоторые наиболее широко обсуждаемые группы проблем.

Дискриминация

Люди, имеющие счета в банках и доступ к банковским услугам, таким как онлайн-платежи или кредиты, оказываются в более привилегированном положении. По данным Всемирного банка, около 38% населения мира не имеет банковских счетов (см. веб-ресурс 6 в приложении В). Цифры постепенно улучшаются, но многие люди по-прежнему могут использовать только наличные.

Без банковского счета и основных банковских услуг, таких как онлайн-платежи, люди не могут распространить свой бизнес за пределы своих стран и регионов. Продавец не может предлагать товары или услуги в Интернете для увеличения своей клиентской базы. Человеку, живущему в сельской местности, возможно, придется провести в дороге полдня, чтобы оплатить счет за коммунальные услуги или пополнить свой мобильный счет.

ПРОБЛЕМЫ

☐ Дискриминация

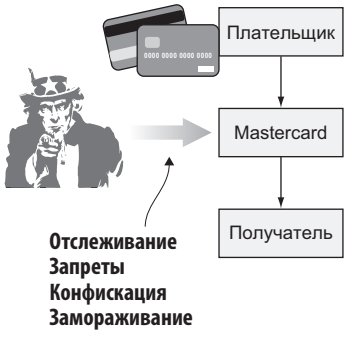
Такая дискриминация в отношении людей, имеющих и не имеющих банковские счета, обусловлена несколькими факторами:

- для некоторых банковские услуги слишком дороги;
- чтобы пользоваться банковскими услугами, нужны документы, например удостоверение личности, которого у многих нет;
- в банковских услугах может быть отказано людям с определенными политическими взглядами или тем, кто ведет определенный бизнес. Людям также может быть отказано в обслуживании из-за этнической принадлежности, национальности, сексуальных предпочтений или цвета кожи.

Проблемы конфиденциальности

Говоря об электронных платежах, например, с использованием кредитных карт или банковских переводах, нельзя не упомянуть о проблемах с конфи-

денциальностью, характерных для традиционных денег. Сотрудники банков легко могут:



- отслеживать платежи;
- запрещать платежи;
- замораживать активы;
- конфисковывать средства со счетов.

Вы можете сказать: «Мне нечего скрывать, а правительству нужны эти инструменты для борьбы с преступностью». Проблема в том, что никто не знает, как правительство будет выглядеть через пять лет и как это правительство определяет преступность. Новые законы — часто результат выборов. После следующих выборов ваше правительство может принять закон, позволяющий замораживать средства граждан, разделяющих вашу политическую точку зрения. В некоторых странах это уже происходит.

ПРОБЛЕМЫ

- ☐ Дискриминация
- ☐ Проблемы конфиденциальности

Есть много примеров, когда власть использовалась, чтобы лишить кого-либо возможности совершать сделки. Например, в 2010 году попала под санкции некоммерческая организация WikiLeaks. Тогда, после давления со стороны правительства США на крупные платежные сети Visa и Mastercard (см. веб-ресурс 7), были заблокированы все пожертвования через традиционные каналы. В 2013 году мы также видели, как Кипр конфисковал 47,5% всех банковских депозитов, превышающих 100 тысяч евро, в рамках программы финансового спасения (веб-ресурс 8).

Обратите внимание, что эта проблема обычно не затрагивает банкноты и монеты. Пока есть наличные деньги, люди могут торговать свободно и в частном порядке. В некоторых странах, например в Швеции, объем наличности постепенно сокращается, то есть вскоре вы не сможете купить даже жевательную резинку, чтобы где-то не зафиксировалась информация об этой транзакции.

Инфляция

Под *инфляцией* подразумевается снижение покупательной способности денег (рис. 1.8).

Большинство валют подвержено инфляции, одни больше, другие меньше. Например, в 2007–2008 годах зимбабвийский доллар упал примерно на

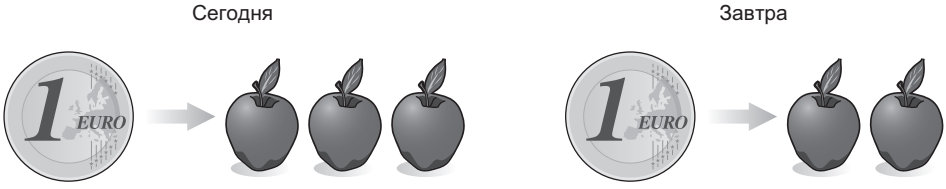


Рис. 1.8. Инфляция

10^{23}%, достигнув пика в 80 миллиардов процентов в месяц, державшегося в течение нескольких месяцев в 2008 году. Это почти 100% среднесуточный уровень инфляции, то есть цены примерно удваивались каждый день.

Подобные экстремальные проявления инфляции называются *гиперинфляцией* и обычно обусловлены ростом денежной массы. Правительства иногда увеличивают денежную массу, стремясь изъять ценности у населения и покрыть такие расходы, как государственный долг, война или социальное обеспечение. При чрезмерном использовании этого инструмента риск гиперинфляции возрастает многократно.

ПРОБЛЕМЫ

- ☐ Дискриминация
- ☐ Проблемы конфиденциальности
- ☐ Инфляция

Быстрый рост денежной массы, скорее всего, приведет к обесцениванию национальной валюты. Это, в свою очередь, заставит людей обменивать местную валюту на товары или валюты других стран, которые лучше справляются с удержанием цен, что еще больше способствует снижению стоимости местной валюты. Ситуация может обостриться до крайности, как в Зимбабве. Результат катастрофичен для людей, поскольку они видят, что их сбережения превращаются практически в ничто. В табл. 1.1 перечислены примеры недавних гиперинфляций.

Таблица 1.1. Некоторые примеры недавних гиперинфляций. Источник: Википедия

Страна	Год	Наибольшая инфляция в месяц (%)
Зимбабве	2007–2008	$4{,}19 \times 10^{16}$
Югославия	1992–1994	313×10^{6}
Перу	1990	397
Украина	1992–1994	285
Венесуэла	2012–	120

Зимбабве показала пример одного из самых экстремальных случаев инфляции за всю историю, но даже сегодня некоторые страны страдают от очень высокой инфляции. Одна из них — Венесуэла, где в 2016 году ее валюта, боливар, претерпела инфляцию в 254%, а в 2017 году — около 1088%. В 2018 году прогнозируется ошеломляющий уровень инфляции в 1 370 000%.

Границы

Перемещать ценности через национальные границы, используя национальную (*фиатную*) валюту, сложно, дорого, а иногда даже незаконно. Если вы решите отправить тысячу шведских крон (SEK) из Швеции человеку на Филиппинах, то сможете воспользоваться услугами, например, Western Union. Когда я исследовал этот вопрос, 1000 шведских крон стоили 5374 филиппинских песо (PHP), или 109 долларов США (табл. 1.2).

Таблица 1.2. Стоимость перевода 5374 филиппинских песо из Швеции на Филиппины

Отправка из	Получение в	Получено фактически	Комиссионные	Комиссионные (%)
Банк	Банк	5109 PHP	265 PHP	4,9%
Банк	Наличные	4810 PHP	564 PHP	10,5%
Кредитная карта	Наличные	4498 PHP	876 PHP	16,3%

Если у получателя есть банковский счет, открытый для получения международных денежных переводов, вы сможете обойтись комиссией 4,9%. Но если получатель ограничен возможностью получать только наличные, тогда комиссия может удвоиться, а то и утроиться до 10,5% или 16,3%, в зависимости от скорости перевода и удобства его получения.

В отличие от международных переводов, перемещение фиатной валюты в пределах государства обычно удобнее. Например, наличные можно передать непосредственно получателю или перевести с помощью какого-либо мобильного приложения, созданного специально для национальной валюты. Пока вы остаетесь в пределах одной страны и одной валюты, фиатные деньги обычно довольно удобны в обращении.

ПРОБЛЕМЫ

- ☐ Дискриминация
- ☐ Проблемы конфиденциальности
- ☐ Инфляция
- ☐ Границы

Подход, предлагаемый технологией Биткоин

Биткоин предлагает модель, принципиально отличающуюся от модели традиционных финансовых институтов. Рассмотрим основные различия.

Децентрализация

Вместо централизованной организации, например Федеральной резервной системы США, контроль биткоинов осуществляется тысячами компьютеров, или узлов. Ни один узел или группа узлов не имеет больше привилегий или обязательств, чем любой другой. Это равенство между узлами делает систему Биткоин *децентрализованной*, в противовес централизованным системам, таким как банки или поисковая система Google (рис. 1.9).

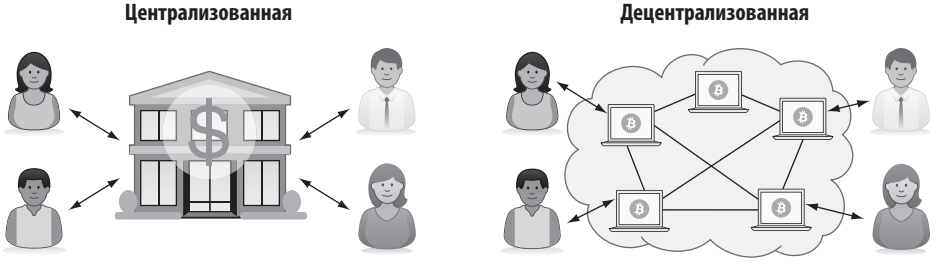


Рис. 1.9. Централизованная и децентрализованная услуги

В централизованной системе услуга контролируется одним субъектом, например банком. Этот субъект может решать, кто получит услугу и что ему разрешено делать. Например, онлайн-видеосервис может предлагать видеофильмы только людям, проживающим в определенном географическом местоположении.

РЕШЕННЫЕ ПРОБЛЕМЫ

- ☑ Дискриминация
- ☑ Проблемы конфиденциальности
- ☐ Инфляция
- ☐ Границы

В децентрализованной системе с несколькими тысячами узлов, разбросанными по всему миру, очень сложно контролировать, кто и как использует систему. Независимо от того, где они проживают, кем являются или кому отправляют деньги, система Биткоин

будет одинаково относиться ко всем пользователям. В системе Биткоин нет центрального органа управления, который можно использовать для цензуры платежей, отказа в обслуживании или конфискации средств.

Как уже отмечалось, система Биткоин не ограничивает доступ к своим услугам, а значит, вам не нужно ни у кого спрашивать разрешения на участие в ней. Любой, у кого есть компьютер и подключение к Интернету, может настроить узел Биткоин и играть активную роль в сети — никаких вопросов не задается и никакой регистрации не требуется.

Изменение правил Биткоин практически невозможно без общего согласия. Если узел не подчиняется правилам, остальные узлы его просто игнорируют. Например, одно из правил гласит, что денежная масса в системе Биткоин ограничена 21 миллионом биткоинов. Этот предел почти невозможно изменить из-за децентрализации; никто не сможет, угрожая или давая взятки, изменить правила.

Ограниченная денежная масса

РЕШЕННЫЕ ПРОБЛЕМЫ

- ☑ Дискриминация
- ☑ Проблемы конфиденциальности
- ☑ Инфляция
- ☐ Границы

Поскольку денежная масса в системе Биткоин не может превысить 21 миллиона биткоинов, люди могут быть уверены, что, владея 1 биткоином, они *всегда* будут владеть, по крайней мере, одной 21-миллионной частью всех биткоинов. Эта особенность не встречается ни в одной фиатной валюте, где решения об эмиссии очень часто принимаются компанией или государством. Биткоин устойчив к высокой инфляции, потому что нет возможности увеличить денежную массу по чьему-либо желанию.

Денежная масса в системе Биткоин пока не достигла предела. Согласно *заранее установленному* графику, она растет с уменьшающейся скоростью и в конечном счете прекратит расти примерно к 2140 году (рис. 1.10).

На момент написания этих строк в обращении находилось около 17 миллионов биткоинов, а текущий ежегодный прирост составлял примерно 4%. Этот прирост уменьшается вдвое каждые четыре года.

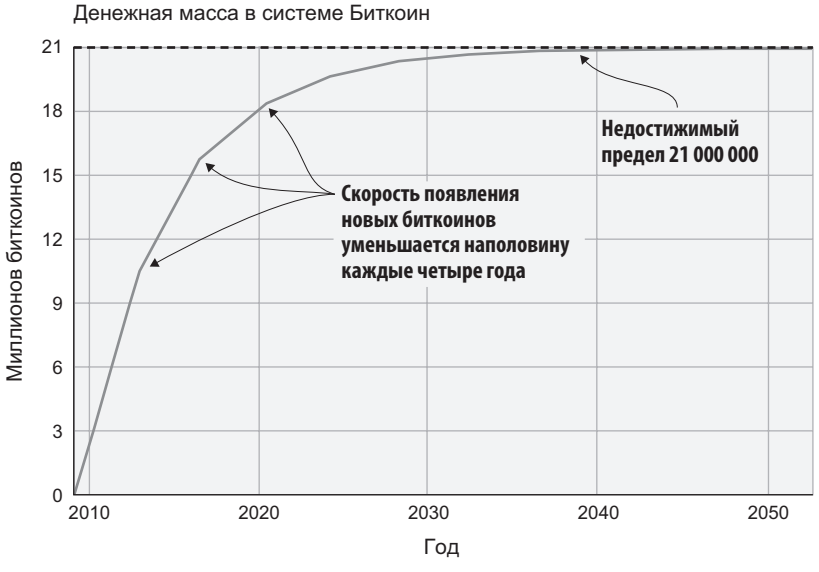


Рис. 1.10. С течением времени количество биткоинов достигнет своего предела, примерно равного 21 миллиону. Увеличение в последние 100 лет перед 2140 годом будет едва заметно

РЕШЕННЫЕ ПРОБЛЕМЫ

- ☑ Дискриминация
- ☑ Проблемы конфиденциальности
- ☑ Инфляция
- ☑ Границы

Отсутствие границ

Поскольку Биткоин — это система, управляемая обычными компьютерами, подключенными к Интернету, она имеет такой же глобальный характер. То есть любой человек, имеющий подключение к Интернету, может отправлять деньги другим людям по всему миру, как показано на рис. 1.11.

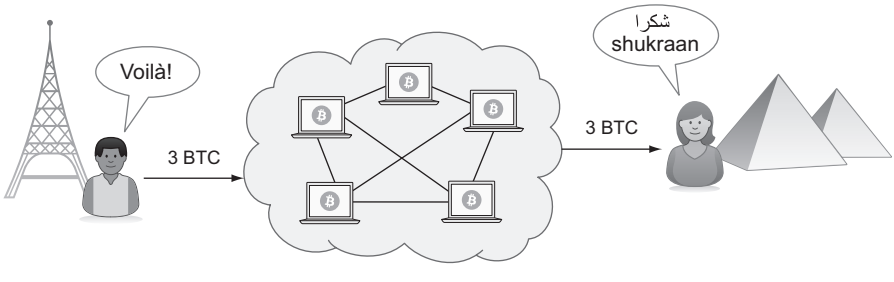


Рис. 1.11. Биткоин не имеет границ

Нет никакой разницы между отправкой биткоина кому-то находящемуся в одной комнате с вами или на другом континенте. Суть от этого не меняется: деньги отправляются непосредственно получателю, который видит платеж почти мгновенно. Получатель может быть *уверен*, что не более чем через час деньги будут принадлежать ему. Как только деньги осядут в кошельке, перевод нельзя отменить без согласия получателя.

Где можно использовать биткоины?

К настоящему моменту мы затронули несколько общих вариантов использования биткоинов. В этом разделе мы подробнее рассмотрим эти и некоторые другие варианты. Трудно предсказать, какие варианты мы увидим в будущем, поэтому будем придерживаться того, что известно сейчас.

Сбережение

Одна из интересных особенностей биткоинов заключается в том, что безопасность хранения денег зависит от безопасности хранения набора закрытых ключей: секретных данных, которые понадобятся, когда вы захотите потратить свои деньги. Вы сами выбираете, как хранить свои закрытые ключи. Можете записать их на бумаге или хранить в электронном виде с помощью мобильного приложения, чтобы упростить к ним доступ. Также можете запомнить свои личные ключи. Эти ключи — все, что нужно, чтобы тратить деньги. Держите их в тайне.

Сбережение — привлекательный вариант использования биткоинов. Самый простой способ сберечь деньги — создать закрытый ключ, записать его на листе бумаги и запереть в сейфе. Этот лист бумаги теперь является вашим сберегательным счетом, вашим сберегательным кошельком. Затем вы можете перевести биткоины в свой кошелек. Пока закрытый ключ хранится в безопасности, ваши деньги тоже будут пребывать в безопасности. На выбор есть множество различных схем сбережения, и вы легко найдете приемлемый для себя баланс между безопасностью и удобством. Например, вы можете хранить ключи в незашифрованном виде в мобильном телефоне для удобного доступа или хранить их в зашифрованном виде в хранилище с вооруженной охраной.

Трансграничные платежи

Как уже отмечалось, перевод денег из одной страны в другую стоит дорого (скажем, 15%), особенно если деньги переводятся в бедную страну, а у получателя нет банковского счета. В последнее время растет популярность использования биткоинов для обхода этой дорогой и медлительной устаревшей системы. Обычно дешевле обменять шведские кроны на биткоины в Швеции, переслать биткоины на Филиппины, а затем обменять биткоины на филиппинские песо.

Некоторые компании предлагают подобную услугу: вы платите шведские кроны компании, а компания выплачивает деньги филиппинскими песо вашему другу (рис. 1.12). Вы даже не узнаете, что за кулисами используются биткоины. Такие компании обычно берут несколько процентов за услугу, но это все равно дешевле, чем традиционные услуги по переводу денег.

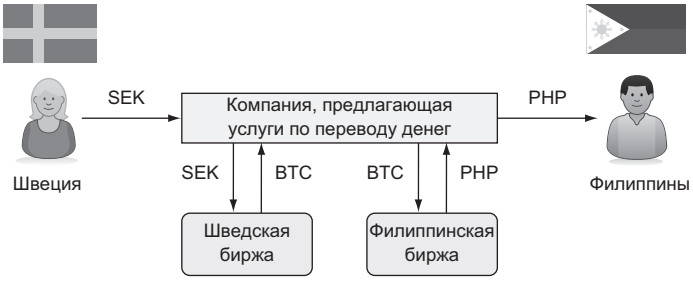


Рис. 1.12. Компания, предлагающая услугу денежных переводов, использует биткоины для передачи денег из Швеции на Филиппины

Конечно, если получатель может свободно использовать биткоины в своей стране, потребность в посреднике, берущем деньги за услуги, отпадает сама собой. Вы можете сами послать биткоины своему другу напрямую. В этом вся прелесть системы Биткоин. Биржи и другие обслуживающие компании — это всего лишь мосты, соединяющие старый мир с новым миром Биткоин.

Покупки

Наиболее очевидный вариант использования системы Биткоин — это покупки. Отсутствие границ и безопасность Биткоин делают ее идеальной для онлайн-платежей за товары и услуги.

Выполняя традиционные онлайн-платежи, вы отправляете данные своей дебетовой карты продавцу и *надеетесь*, что продавец снимет ровно ту сумму, с которой вы согласились. Вы также *надеетесь*, что продавец предпримет все меры безопасности при обработке данных вашей дебетовой карты. Вполне возможно, что эти данные он сохранит в базе данных. Только представьте: при оплате каждой покупки дебетовой картой информация о карте будет сохраняться в базе данных продавца. Вполне вероятно, что *одна* из баз данных будет взломана, а данные украдены. Чем больше продавцов сохранит ваши данные, тем выше риск.

В системе Биткоин эта проблема отсутствует, потому что вы не посылаете конфиденциальную информацию продавцу или кому-либо еще. Вы переводите сумму, которую согласились перевести, и больше ничего.

Спекуляции на курсах

Мир полон людей, желающих быстро разбогатеть. Биткоин может привлекать их *волатильностью* цен или динамикой курса. Глядя на историю цен на биткоины, изображенную на рис. 1.13, представляется заманчивым купить биткоины по низкой цене и продать по высокой.

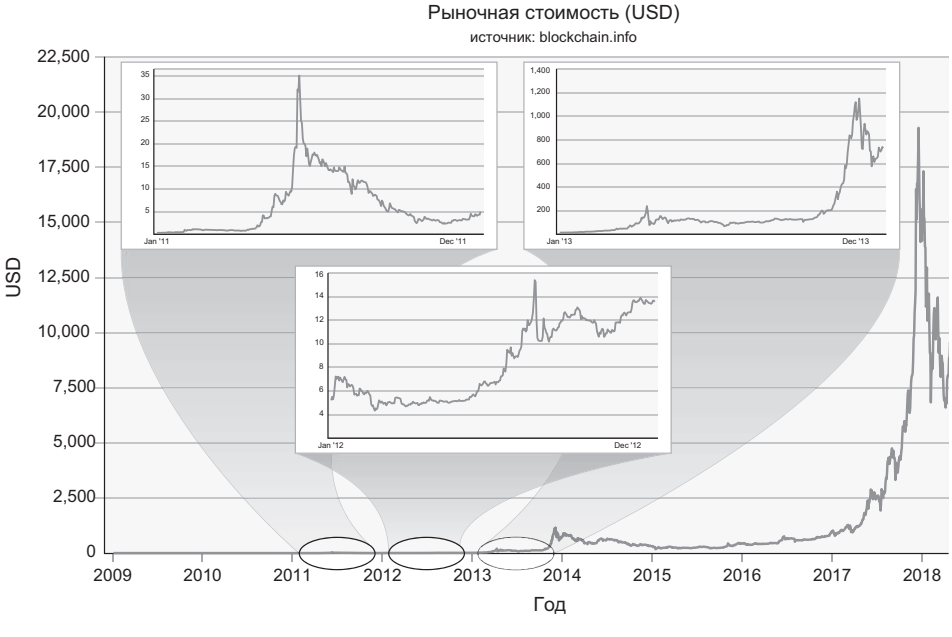


Рис. 1.13. Цены в долларах США (USD) за всю историю Биткоин

В ноябре 2013 года за несколько недель цена биткоина поднялась с примерно 100 долларов США до более чем 1100. Это явно был так называемый *пузырь*, в котором люди боялись упустить значительную прибыль и покупали, повышая цену до тех пор, пока она в конечном итоге не начала снова падать. Падение до 50% от пикового значения было столь же быстрым, как и рост. Та же картина повторилась в конце 2017 года, но с большим размахом. Это случалось уже много раз. Подобные колебания редко бывают вызваны какими-либо конкретными новостями или технологическим прогрессом и чаще возникают из-за спекуляций. Игра на курсах может быть захватывающей, если вы можете позволить себе проиграть, но она больше похожа на лотерею, чем на реальный заработок. Иногда правительство или крупная корпорация делает негативное заявление о Биткоин, порождающее страх на рынке, но эти события, как правило, оказывают ограниченное влияние на стоимость самих монет.

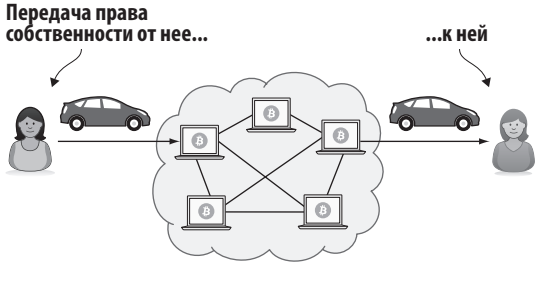
Волатильность цены на биткоин противоречит утверждениям о том, что он обладает неинфляционным свойством; падение рыночной стоимости на 50% выглядит вполне себе инфляционным. Биткоин все еще относительно нов, и множество краткосрочных спекуляций вызывает волатильность. Но биткоин растет, и все больше людей и организаций начинают использовать его для хранения своих богатств, поэтому он, вероятно, стабилизируется в долгосрочной перспективе, и его дефляционное свойство проявится со временем.

Безвалютное использование

Биткоин — цифровая валюта, но эту форму денег можно использовать не только как деньги. В этом разделе описываются два основных способа использования, но есть и другие, в том числе еще не изобретенные.

Право собственности

Биткоин позволяет внедрять в платежи небольшие фрагменты данных. Этими данными может быть, например, номер кузова автомобиля. Когда автомобиль покидает завод, производитель может перевести небольшую сумму в бит-



коинах новому владельцу автомобиля, указав номер шасси. Этот платеж будет представлять передачу права собственности на автомобиль.

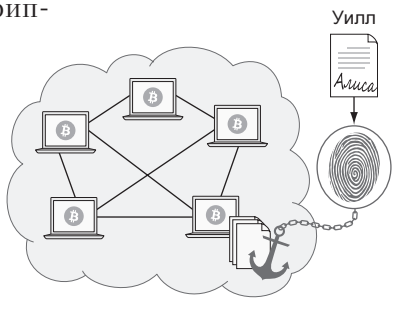
Платежи в биткоинах являются открытыми записями, но они никак не связаны с людьми. Они привязаны к длинным цепочкам цифр, называемым *открытыми ключами*, которые подробно описываются в главе 2. Производитель автомобилей может опубликовать свой открытый ключ на своем веб-сайте, в газетах и в рекламных объявлениях, чтобы связать ключ с личностью производителя. После этого любой сможет убедиться, что производитель передал право собственности на автомобиль новому владельцу. Новый владелец сможет доказать, что владеет автомобилем, с помощью своего закрытого ключа, соответствующего открытому ключу, посредством которого производитель передал право собственности.

Новый владелец может продать автомобиль кому-то другому и передать право собственности, отправив ту же сумму в биткоинах, которую получил от производителя, на открытый ключ нового владельца. Широкая публика сможет проследить, как менялось право собственности на автомобиль от производителя, через открытый ключ каждого владельца.

Доказательство существования

Используя тот же метод хранения данных в биткоин-платеже для передачи права собственности на автомобиль, можно доказать, что документ существовал до определенного момента времени.

Цифровой документ имеет *сигнатуру*: криптографический хеш, который любой может вычислить по документу. Создание другого документа с той же сигнатурой практически невозможно. Эту сигнатуру можно прикрепить к платежу в биткоинах. Куда уходят деньги, не имеет значения; важно, что сигнатура сохранится в блокчейне Биткоин. Таким способом можно «привязать» документ к блокчейну.



Платежи в биткоинах являются открытыми записями, поэтому любой сможет убедиться в существовании документа до момента оплаты, рассчитав сигнатуру документа и сравнив его с сигнатурой, хранящейся в блокчейне.

Как оцениваются биткоины?

Как рассказывалось в разделе «Игра на курсах», цена биткоина может сильно колебаться. Но как эта цена определяется? Существует несколько бирж, торгующих биткоинами, большинство из которых проводят торги в Интернете. Они напоминают фондовые биржи, где встречаются интересы пользователей, желающих продать и купить биткоины.

Разные рынки могут иметь разные цены, в зависимости от спроса и предложения. Например, в таких странах, как Венесуэла, где правительство пытается помешать развитию рынка биткоинов, предложение невелико. А спрос высок, потому что люди стремятся избавиться от своей гиперинфляционной валюты. Эти факторы приводят к росту цен на биткоины на этом рынке, по сравнению, например, с рынками США и Европы, где люди могут торговать более свободно.

В каких случаях биткоины не могут использоваться

Биткоин имеет много достоинств, но его можно использовать не во всех ситуациях. По крайней мере пока.

Микроплатежи

Транзакция в системе Биткоин обычно должна включать комиссию за обработку. Величина комиссии зависит не от суммы, а от размера транзакции в байтах, потому что цена обработки транзакции в сети Биткоин в основном зависит от ее размера (в байтах). Транзакции с большой суммой ничуть не больше (в байтах) транзакции с маленькой суммой, поэтому комиссия за обработку обеих транзакций примерно одинакова. Комиссия за транзакцию также зависит от спроса и предложения на доступное пространство в блокчейне. Блокчейн не может обрабатывать более 12 Мбайт транзакций

в час, поэтому иногда майнеры вынуждены определять приоритеты для транзакций. Согласие на более высокую комиссию, вероятно, даст вашей транзакции более высокий приоритет.

Если комиссия составляет значительную долю фактического платежа, который вы хотите сделать, платить обычными биткоин-транзакциями становится невыгодно (табл. 1.3).

Таблица 1.3. Обоснованность разных комиссий

Сумма платежа	Комиссия	Комиссия (%)	Обоснованно?
2 BTC	0,003 BTC	0,15%	Да
0,002 BTC	0,001 BTC	50%	Вероятно, нет
0,001 BTC	0,005 BTC	500%	Нет

Но на основе Биткоин строятся новые многообещающие технологии. Одним из примеров является Lightning Network (сеть-молния) — протокол, который позволяет мгновенно и дешево проводить микроплатежи в долях биткоина. С помощью Lightning Network можно провести платеж всего на 100 сатоши (1 сатоши = 0,00000001 BTC) за комиссию, равную 1 сатоши.

Мгновенные платежи

Для подтверждения платежей в Биткоин требуется время. Получатель видит платеж немедленно, но не может быть уверенным в нем, пока сеть Биткоин не подтвердит его, на что обычно требуется около 20 минут. Доверять неподтвержденной транзакции рискованно; отправитель может *дважды потратить* биткоины, отправив те же самые биткоины в другой транзакции на другой биткоин-адрес — например, на адрес отправителя.

Необходимость ждать подтверждения может сделать затруднительным использование биткоинов в обычной торговле, потому что клиенты не захотят ждать 20 минут, прежде чем получат свой кофе. Это едва ли станет проблемой для онлайн-магазинов, потому что такой магазин легко может подождать 20 минут, прежде чем отправить товар покупателю; но в некоторых онлайн-службах, например, взимающих плату за просмотр, задержка в ожидании подтверждения может сделать затруднительным использование биткоинов.

Это ограничение тоже может быть преодолено с помощью систем, построенных на основе Биткоин, таких как Lightning Network, особенно когда сумма платежа мала.

Сбережения, потеря которых недопустима

БЕЗОПАСНОСТЬ БИТКОИН

Вы и только вы отвечаете за безопасность своих биткоинов. Будьте осторожны!

Биткоин, пожалуй, самая надежная валюта из имеющихся, но она все еще находится в зачаточном состоянии. С биткоинами *можно* попасть в очень плохую ситуацию, например, если:

- вы потеряете свой закрытый ключ: без закрытого ключа вы не сможете распоряжаться своими деньгами;
- ваши закрытые ключи будут украдены;
- правительство вашей страны начнет преследовать пользователей Биткоин введением уголовного наказания или с использованием других силовых методов;
- цена биткоинов резко упадет из-за слухов или спекуляций;
- в программном обеспечении Биткоин обнаружатся уязвимости;
- обнаружатся недостатки в криптографических процедурах, используемых в Биткоин.

Хотя все эти ситуации *возможны*, большинство из них маловероятно. События в этом списке перечислены в порядке от наиболее вероятных к менее вероятным. Всегда трезво взвешивайте риски и выбирайте соответствующие меры безопасности. Эта книга поможет вам понять риски и расскажет, как обезопасить деньги.

Другие криптовалюты

Эта книга рассказывает о Биткоин, но существует несколько других *криптовалют* и постоянно появляются новые. Криптовалюты, отличные от Биткоин, часто называют *альткоинами* (alt-coins), то есть *альтернативными монетами*. В табл. 1.4 я перечислил несколько альткоинов вместе с их назначением и рыночной капитализацией. Рыночная капитализация — это произведение денежной массы (количества монет) и текущей рыночной цены одной монеты. Обратите внимание, что рыночная капитализация, скорее всего, сильно изменится к тому времени, когда вы будете читать эти строки. Я включил эту информацию только для того, чтобы дать вам представление о положении Биткоин среди других криптовалют.

Рекомендую познакомиться с этими криптовалютами, потому что все они предлагают новые и интересные возможности сверх тех, что предлагает Биткоин. Существуют сотни других альткоинов. Некоторые из них, подобно

перечисленным в табл. 1.4, предлагают уникальные возможности, недоступные в Биткоин, а другие — почти ничего инновационного. Некоторые альткоины даже могут создаваться с мошенническими целями. Поэтому будьте бдительны. Любой может создать альткоин, взяв существующее программное обеспечение поддержки криптовалют и модифицировав его под свои потребности.

Таблица 1.4. Рыночная капитализация некоторых криптовалют на 11 ноября 2018 г.

Валюта	Назначение	Рыночная капитализация (миллиардов долларов США)
bitcoin	Глобальные деньги, добавлено для сравнения	111
ethereum	Выполнение программ на абстрактном децентрализованном компьютере	22,4
MONERO	Безопасность	1,7
ZCASH	Безопасность	0,8
namecoin	Система имен; дополняет систему доменных имен (DNS)	0,008

Допустим, Шейла решила создать альткоин под названием Ваукоин (Wowcoin). Она берет программное обеспечение Биткоин и уменьшает максимальную денежную массу с 21 000 000 до 11 000 000 монет. В момент запуска Ваукоин Шейла будет единственным пользователем, потому что пока никто не использует ее альткоины. Если она захочет, чтобы ее криптовалюта Ваукоин приобрела реальную ценность, ей придется убедить других начать использовать ее. Если она не предложит ничего инновационного, ей будет трудно привлечь других, потому их вполне устраивает Биткоин. Все остальные используют Биткоин, так зачем им использовать Ваукоин? Эту ситуацию можно сравнить с запуском нового Интернета, который вы называете Ваунет. Люди в Ваунете не смогут пользоваться услугами из Интернета. И наоборот, люди в Интернете не смогут использовать услуги из Ваунета. Так зачем кому-то использовать Ваунет? Мы называем это *сетевым эффектом* (рис. 1.14) — люди стремятся туда, где находятся другие люди.

Хотя некоторые альткоины действительно довольно интересны, трудно сказать, какой из них выживет в долгосрочной перспективе. Кроме того, выбор каких-либо альткоинов для освещения в этой книге был бы в значительной мере произвольным. Поэтому я сосредоточусь исключительно на Биткоин.

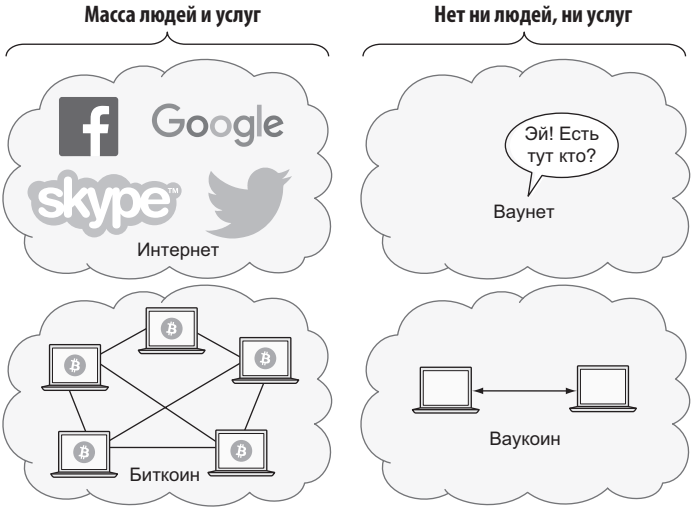


Рис. 1.14. Сетевой эффект

Итоги

- Биткоин — это глобальные деньги, которые может использовать любой имеющий подключение к Интернету.
- Много разных людей используют Биткоин, включая продавцов и трейдеров, для достижения самых разных целей, например для оплаты товаров и услуг, денежных переводов и сбережения накоплений.
- Проверку и учет всех платежей проводит сеть компьютеров — сеть Биткоин.
- Транзакция выполняется в несколько этапов: отправка, проверка, добавление в блокчейн (цепочку блоков) и уведомление кошельков получателя и отправителя.
- Биткоин решает проблемы с инфляцией, границами, дискриминацией и конфиденциальностью, обеспечивая ограниченное предложение, децентрализацию и отсутствие границ.
- Кроме Биткоин существует несколько альтернативных криптовалют: Ethereum, Zcash и Namecoin.
- Ценность (крипто)валюты растет с увеличением числа использующих ее. Это называется сетевым эффектом.

2 Криптографические хеш-функции и цифровые подписи

Эта глава охватывает следующие темы:

- ✓ создание простой денежной системы: жетоны на булочки;
- ✓ знакомство с криптографическими хеш-функциями;
- ✓ аутентификация платежей с использованием цифровых подписей;
- ✓ сохранение секретов в секрете.

В этой главе мы рассмотрим основные темы, необходимые для понимания остальной части книги. Мы рассмотрим простую платежную систему, которую сможем улучшить с помощью технологии Биткоин. Когда мы достигнем главы 8, эта простая система разовьется в то, что мы называем Биткоин.

Во второй части этой главы я познакомлю вас с криптографическими хеш-функциями. Они играют очень важную роль в Биткоин, поэтому нужно понять их, прежде чем изучать что-либо еще. Вы увидите, как с помощью криптографической хеш-функции можно убедиться, что файл не изменялся после последней проверки.

В оставшейся части главы мы займемся решением проблемы *самозванца*: злоумышленника, притворяющегося кем-то другим, чтобы платить деньги со счета этого другого. Мы решим ее, введя цифровые подписи (рис. 2.1) в простую систему.

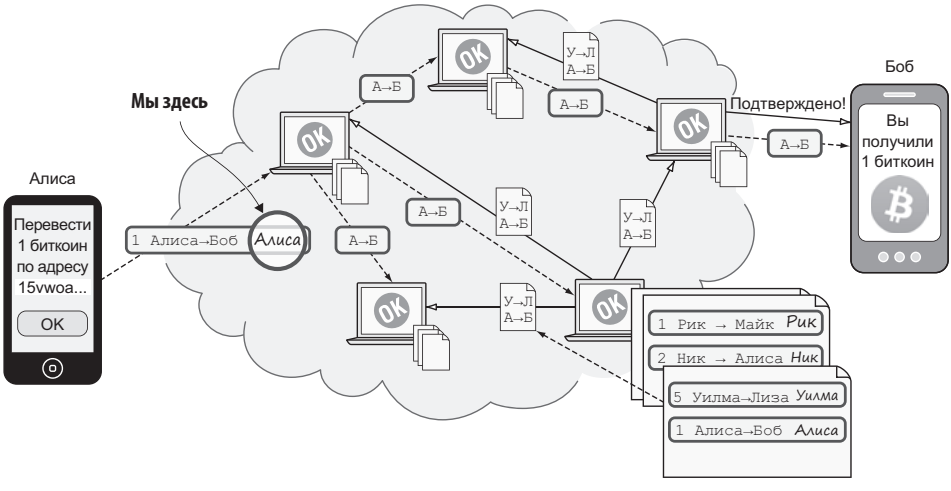


Рис. 2.1. Цифровые подписи в Биткоин

Электронная таблица учета жетонов на булочки

Предположим, в здании, где вы работаете, есть кафе. Вы и коллеги используете электронную таблицу для учета *жетонов на булочки* (рис. 2.2), которые обозначим символами CT[1]. Эти жетоны можно обменять в кафе на булочки.

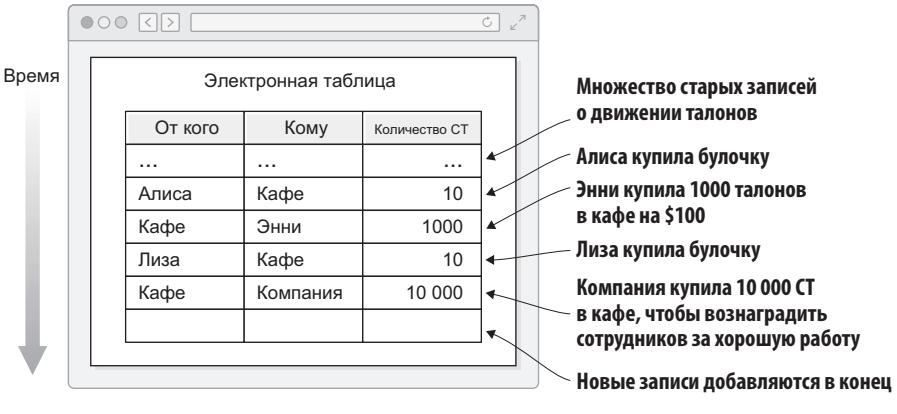


Рис. 2.2. В электронной таблице учета жетонов есть столбцы, определяющие отправителя и получателя, и есть столбец с числом переданных жетонов. Новые записи о движении жетонов добавляются в конец таблицы

[1] От *англ.* cookie tokens — жетоны на булочки. — *Примеч. пер.*

БИТКОИН, ВАЛЮТА

Жетон на булочки соответствует биткоину, единице валюты Биткоин. Первую реальную стоимость биткоин приобрел в 2010 году, когда кто-то купил две пиццы за 10 000 BTC. В ноябре 2018-го за эти деньги можно было бы купить 6 000 000 пицц.

Лиза хранит эту электронную таблицу на своем компьютере. Всем сотрудникам в офисе, кроме Лизы, она доступна только для чтения. Лиза пользуется очень высоким доверием. Все доверяют ей. У нее есть полный доступ к таблице, и она может сделать с ней все, что захочет. Вы и все остальные можете только просматривать электронную таблицу.

Всякий раз, когда Алиса хочет булочек, она просит Лизу, которая работает рядом с кафе, перевести 10 CT от Алисы в кафе. Лиза знает, кто такая Алиса, и, заглянув в электронную таблицу, может убедиться, что у нее достаточно жетонов; она выполнит поиск по имени «Алиса», сложит все числа в столбце «Кому» в записях с именем Алисы и вычтет из суммы все числа в столбце «От» в записях с именем Алисы. На рис. 2.3 изображены все результаты поиска; в таблице есть три записи с именем Алисы.

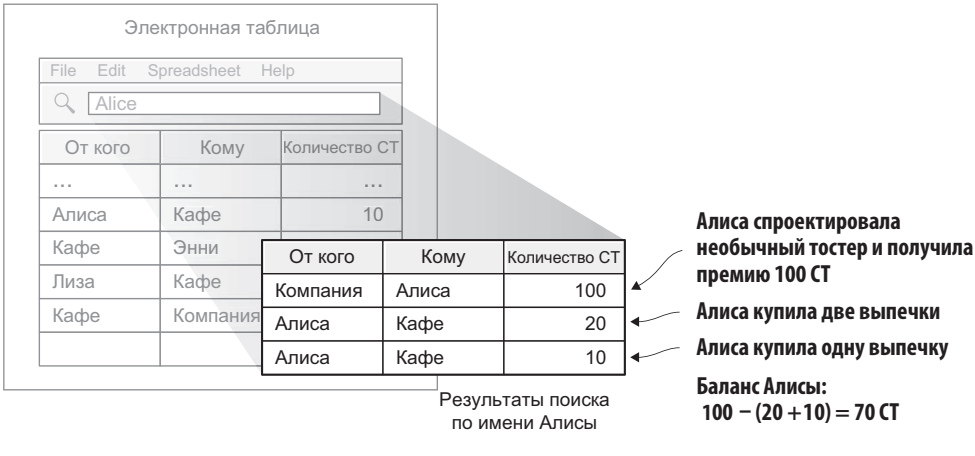


Рис. 2.3. Лиза вычисляет баланс Алисы. Сумма полученных ею жетонов равна 100, а сумма потраченных равна 30. Соответственно Алиса имеет 70 CT

Выяснив, что на своем балансе Алиса имеет 70 CT (достаточно, чтобы перевести 10 CT в кафе), Лиза добавит запись в конец таблицы (рис. 2.4).

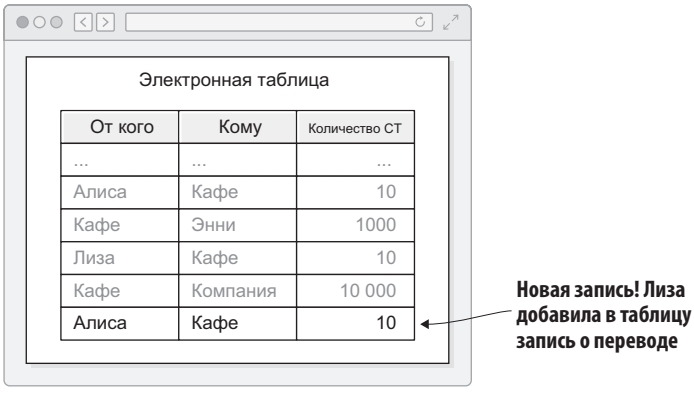


Рис. 2.4. Лиза добавила запись о платеже за булочки. Запись добавлена в конец таблицы учета жетонов

Сотрудник кафе заметит новую запись в таблице и передаст булочки Алисе.

Когда жетоны закончатся, вы сможете купить их за доллары у кого-то, кто согласится продать вам немного, например у Энни или в кафе, по цене, о которой вы договоритесь. Лиза добавит в таблицу соответствующую запись.

ЗАРАБОТАЙ ИХ

Также есть возможность получить часть заработной платы жетонами.

Лиза пообещала никогда не удалять и не изменять что-либо в электронной таблице, а только добавлять новые записи. Все, что попало в таблицу, остается неизменным!

Лиза выполняет важную работу, обеспечивая безопасность этой денежной системы, и ежедневно получает за это 7200 новых жетонов (рис. 2.5). Каждый день она добавляет в таблицу новую строку, записывая на свой счет 7200 новых жетонов.

Так создаются все жетоны на булочки в электронной таблице. Первая запись в таблице — это запись с вознаграждением, подобная изображенной на рис. 2.5, создавшая самые первые 7200 СТ. Согласно плану, Лиза будет ежедневно получать вознаграждение в 7200 СТ в течение первых четырех лет, в течение следующих четырех лет вознаграждение будет уменьшено вдвое — до 3600 СТ в день, и так далее, пока вознаграждение не уменьшится до 0 СТ в день.

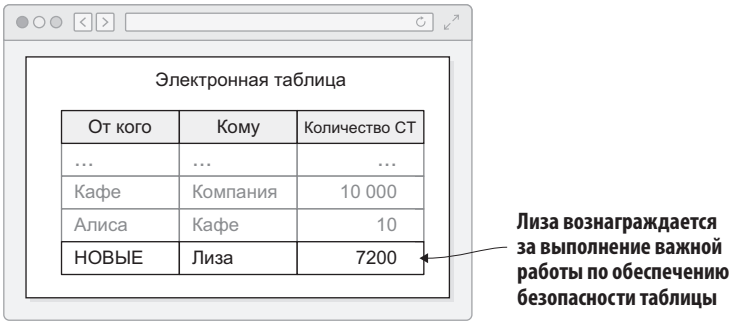
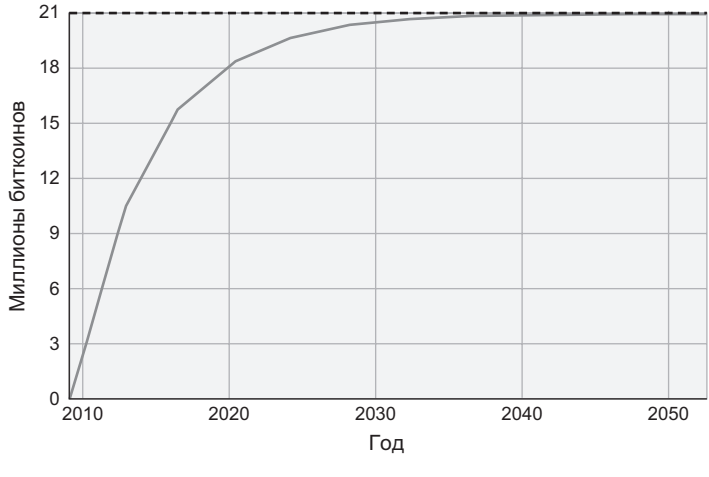


Рис. 2.5. Лиза вознаграждается жетонами на булочки

Пока нет причин волноваться о том, что произойдет, когда сумма вознаграждения достигнет 0 — это вопрос отдаленного будущего. Мы обсудим его в главе 7. Сокращение вдвое каждые четыре года приведет к тому, что общая денежная масса — общее количество жетонов в обращении — приблизится к 21 миллиону CT, но никогда не превысит этого числа.

КРИВАЯ ВЫПУСКА НОВЫХ ЖЕТОНОВ

В системе Биткоин действует тот же график выпуска новых монет, что и в случае с жетонами на булочки. Все новые биткоины создаются как вознаграждение узлам за безопасность консенсусного реестра Биткоин — блокчейна, в точности как Лиза вознаграждается за безопасность электронной таблицы.



Как Лиза распорядится новыми жетонами — это ее дело. Она может продать жетоны или купить на них булочки. Она также может сохранить их на будущее. Система с электронной таблицей работает хорошо, и каждый получает неплохую порцию булочек.

Фактически Лиза выполняет ту же работу, что и майнеры в сети Биткоин. Она проверяет платежи и обновляет консенсусный реестр — электронную таблицу учета жетонов на булочки. В табл. 2.1 поясняется, как идея электронной таблицы соответствует идее Биткоин.

Таблица 2.1. Как ключевые понятия системы с электронной таблицей соотносятся с системой Биткоин

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
Электронная таблица	Блокчейн	Глава 6
Запись в электронной таблице	Транзакция	Глава 5
Лиза	Майнер	Глава 7

Эта таблица будет сопровождать нас на протяжении всей книги. В ней описаны различия между жетонами на булочки и биткоинами. Я буду удалять из нее строки по мере знакомства с понятиями из мира Биткоин. Например, строку «Электронная таблица» я удалю в главе 6, когда расскажу, как для хранения транзакций используется блокчейн. Я также буду добавлять новые строки, представляя новые идеи для системы жетонов, которые отличаются от аналогов в Биткоин.

К концу главы 8 в этой таблице останется только первая строка, описывающая аналогию жетона с биткоином. Это будет означать конец примера с жетонами на булочки, и с этого момента мы будем говорить только о Биткоин.

В табл. 2.2 представлена наша отправная точка в изучении работы Биткоин, которую мы назовем версией 1.0 системы жетонов на булочки в электронной таблице.

Таблица 2.2. Примечания к релизу, жетоны на булочки 1.0

Версия	Особенность	Как реализована
НОВАЯ 1.0	Простая платежная система	Основывается на доверии Лизе и личных знакомствах
	Конечный объем денег	Лиза получает в награду 7200 CT ежедневно; величина вознаграждения уменьшается вдвое каждые четыре года

В каждой главе мы будем добавлять в эту систему много интересных возможностей и выпускать новую версию. Например, в конце этой главы мы выпустим версию 2.0, использующую цифровые подписи для решения проблемы самозванцев. Каждая глава будет приближать нас к конечному результату: системе Биткоин. Но имейте в виду, что Биткоин развивается совсем не так — я использую эту вымышленную систему, только чтобы объяснить каждую важную деталь в отдельности.

Криптографические хеши

Криптографические хеши используются в Биткоин повсеместно. Пытаться изучать Биткоин, не зная, что такое криптографические хеши, это все равно что пытаться изучать химию, не зная, что такое атом.

Криптографический хеш можно сравнить с отпечатками пальцев. Отпечаток левого большого пальца человека не изменяется с течением времени, но крайне сложно найти другого человека с таким же отпечатком левого большого пальца. Отпечаток пальца не раскрывает никакой информации о человеке, кроме самого отпечатка. Рассматривая отпечаток пальца человека, нельзя сказать, какой уровень подготовки в математике тот имеет или какого цвета у него глаза.

Цифровая информация тоже имеет уникальный отпечаток пальца (сигнатуру). Этот отпечаток называется *криптографическим хешем*. Чтобы получить криптографический хеш файла, нужно передать файл компьютерной программе, которая называется *криптографической хеш-функцией*. Предположим, вам понадобилось получить криптографический хеш — отпечаток пальца — вашего любимого изображения кота. Схема на рис. 2.6 иллюстрирует этот процесс.

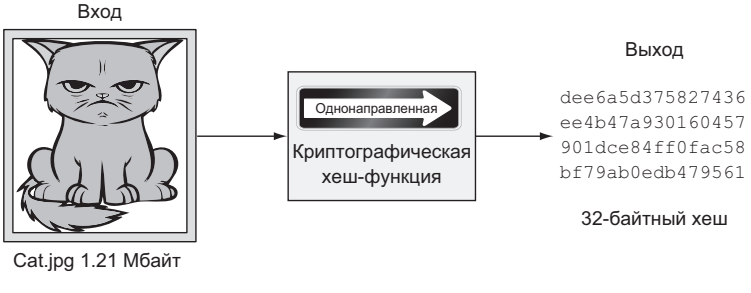
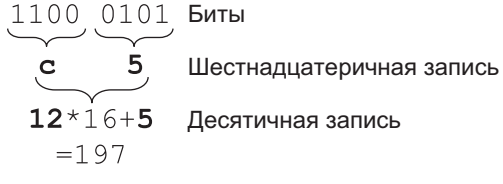


Рис. 2.6. Получение криптографического хеша изображения кота. На вход подается изображение, а на выходе получается большое 32-байтное число

На выходе получается хеш — 256-битное число; 256 бит — это 32 байта, потому что 1 байт состоит из 8 бит. То есть, если сохранить число в файле, его размер составит 32 байта, что очень мало в сравнении с размером изображения кота, составляющим 1,21 Мбайта. В этом примере использовалась криптографическая хеш-функция, которая называется SHA256 (алгоритм безопасного хеширования с 256-битным результатом) и широко используется в Биткоин.

БИТЫ? БАЙТЫ? ШЕСТНАДЦАТЕРИЧНАЯ ЗАПИСЬ?

Бит — это наименьшая единица информации в компьютере. Бит может иметь одно из двух значений: 0 или 1. Подобно лампочке, он может быть включен или выключен. *Байт* — это группа или последовательность из 8 бит и может иметь 256 разных значений. Для изображения чисел в этой книге часто используется *шестнадцатеричная* форма записи. Каждый байт можно записать двумя шестнадцатеричными цифрами — от 0 до f, где a = 10 и f = 15.



Слово *хеш* (hash) означает что-то, что нарезано на мелкие кусочки или перемешано. Это довольно точно описывает принцип действия криптографической хеш-функции. Она получает изображение кота и, используя его как основу, выполняет математические вычисления. На выходе получается большое число — криптографический хеш, — который совсем не похож на изображение кота. Вы не сможете «восстановить» изображение кота, имея только его хеш — *криптографическая хеш-функция является однонаправленной*. На рис. 2.7 показано, что получится, если немного изменить изображение кота и передать его на вход той же криптографической хеш-функции.

В результате получился совершенно другой хеш. Давайте сравним их.

- Прежний хеш:

```
dee6a5d375827436ee4b47a930160457901dce84ff0fac58bf79ab0edb479561
```

- Новый хеш:

```
d2ca4f53c825730186db9ea585075f96cd6df1bfd4fb7c687a23b912b2b39bf6
```

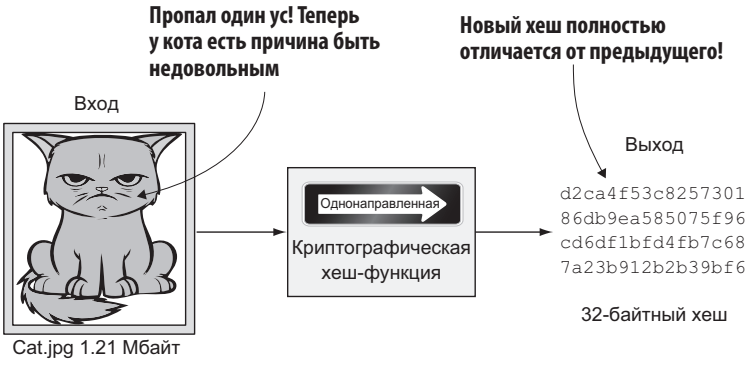


Рис. 2.7. Хеширование измененного изображения кота.
Заметили разницу? Криптографическая хеш-функция точно заметила

Видите, как крохотное изменение в изображении повлекло существенное изменение хеш-значения? Новый хеш имеет совершенно другое значение, но длина хеша всегда одинакова независимо от объема входных данных на входе. Если подать на вход слово «Hello», на выходе все равно получится 256-битный хеш.

Где могут пригодиться криптографические хеш-функции?

Криптографические хеш-функции можно использовать для проверки целостности и обнаружения изменений в данных. Предположим, вы решили сохранить изображение любимого кота на жестком диске вашего ноутбука, но подозреваете, что файл с изображением может быть поврежден. Это может произойти, например, из-за ошибок на диске или атаки хакеров. Как убедиться, что файл не был поврежден?

Для этого сначала нужно вычислить криптографический хеш изображения кота на жестком диске и записать его на листке бумаги (рис. 2.8).

Позже, когда вы захотите посмотреть на изображение, вы можете проверить, изменилось ли оно с того момента, как вы написали хеш на бумаге. Для этого вы снова вычислите криптографический хеш изображения и сравните его с исходным хешем на бумаге (рис. 2.9).

Если новый хеш совпадает с хешем на бумаге, вы можете быть уверены, что изображение не изменилось. Но если хеши не совпадают, изображение кота определенно было подвергнуто изменению.

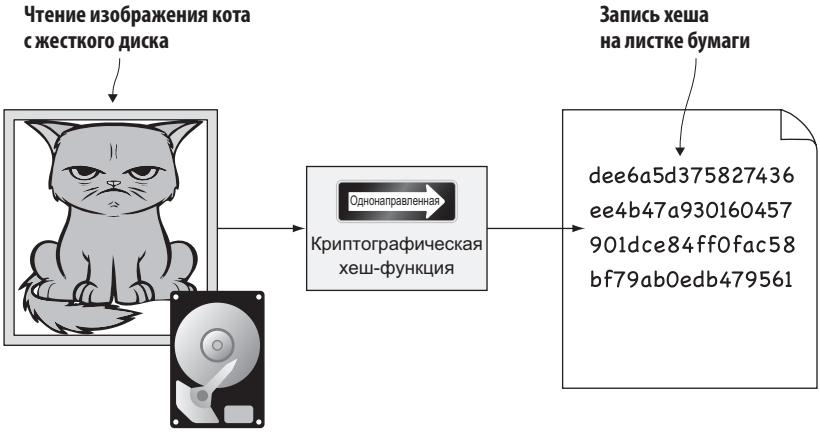


Рис. 2.8. Запись хеша на листке бумаги

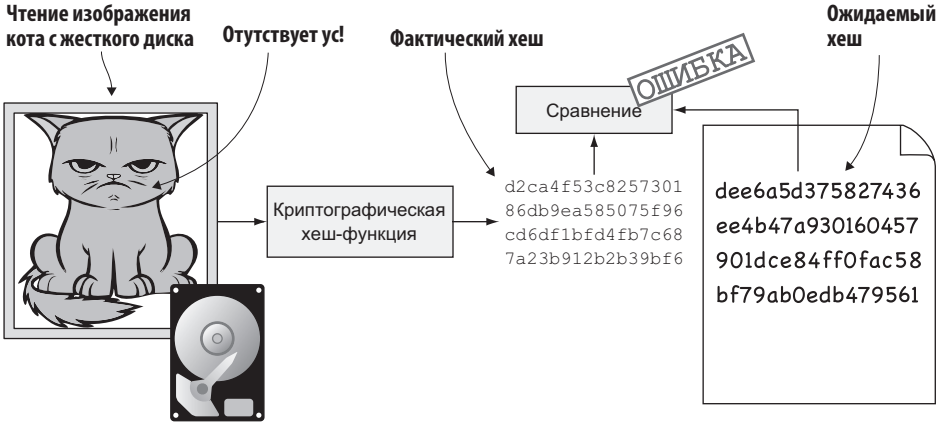


Рис. 2.9. Проверка целостности изображения. В ходе проверки обнаружилось, что изображение было изменено

НАСКОЛЬКО МОЖНО БЫТЬ УВЕРЕННЫМИ В ПРОВЕРКЕ С ПОМОЩЬЮ ХЕША?

Существует небольшая вероятность, что изображение кота изменится, но его хеш останется прежним. Но, как будет показано ниже, эта вероятность настолько мала, что ею можно пренебречь.

Криптографические хеш-функции широко используются в Биткоин для проверки факта неизменности данных. Например, время от времени —

в среднем каждые 10 минут — создается новый хеш всей истории платежей. Если кто-то попытается изменить данные, любой сможет обнаружить подмену, проверив хеш.

Как работают криптографические хеш-функции?

Это довольно сложный вопрос, поэтому я не буду вдаваться в подробности. Но чтобы вы могли понять, как действует криптографическая хеш-функция, мы создадим очень упрощенный ее аналог. Это будет не совсем криптографическая функция, как я объясню позже. Давайте пока просто назовем это хеш-функцией.

Предположим, вы решили получить хеш для файла, содержащего шесть байтов: `a1 02 12 6b c6 7d`. При этом хеш должен быть 1-байтным (8-битным) числом. Вы можете сконструировать хеш-функцию, использовав *сложение по модулю 256*, выполняющее переход к 0, когда результат сложения достигает 256 (рис. 2.10).

МОДУЛЬ

Модуль означает переход в начало, когда результат вычислений превышает некоторое значение. Например:

- 0 mod 256 = 0
- 255 mod 256 = 255
- 256 mod 256 = 0
- 257 mod 256 = 1
- 258 mod 256 = 2

258 mod 256 — это остаток от целочисленного деления 258/256: 258 = 1 × 256 + + 2. Остаток в данном случае равен 2.

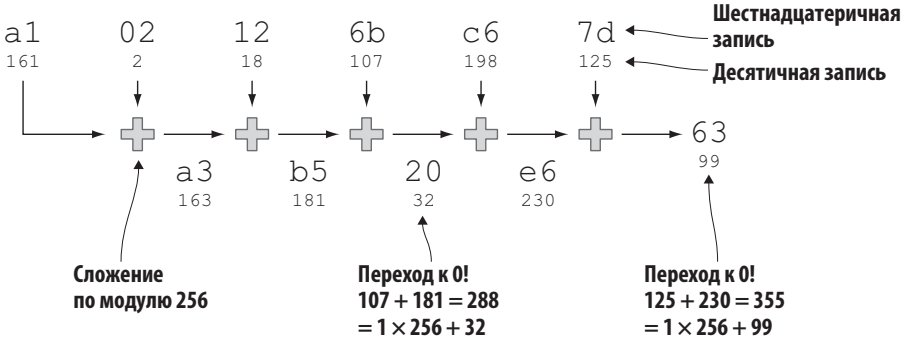


Рис. 2.10. Упрощенная хеш-функция, использующая однобайтное сложение по модулю 256

В результате получается десятичное число 99. Что говорит число 99 об исходной последовательности `a1 02 12 6b c6 7d`? Совсем немного — число 99 выглядит таким же случайным, как и любое другое однобайтное число.

Если изменить входные данные, хеш изменится, хотя есть вероятность, что хеш останется равным 99. В конце концов, эта простая хеш-функция может вернуть лишь 256 разных результатов. В настоящих криптографических хеш-функциях, как те, что мы использовали для хеширования изображения кота, эта вероятность невообразимо мала, в чем вы вскоре убедитесь.

Свойства криптографических хеш-функций

Криптографическая хеш-функция принимает любые цифровые данные, которые называют *исходным образом*, и создает выходные данные фиксированной длины, которые называют *хешем*. В примере с изображением кота на жестком диске исходным образом является изображение кота размером 1,21 Мбайта, а хешем — 256-битное число. Функция будет возвращать один и тот же хеш каждый раз для одного и того же исходного образа. Но если передать ей предварительный образ даже с самыми мелкими изменениями, она почти наверняка вернет совершенно другой хеш. Хеш также обычно называют *дайджестом* (digest).

Давайте посмотрим, какими свойствами обладает типичная криптографическая хеш-функция. Для иллюстрации я возьму SHA256, потому что именно она чаще всего используется в системе Биткоин. Имеются и другие криптографические хеш-функции, но все они обладают одинаковыми базовыми свойствами:

1. **Для одних и тех же исходных данных генерируется один и тот же хеш.**
2. **Если входные данные изменить, на выходе получится другой хеш.**
3. **Хеш всегда имеет один и тот же фиксированный размер. Для SHA256 он равен 256 битам.**
4. **Метод полного перебора — единственный известный способ определить входные данные, дающие в результате заданный хеш.**

Рисунок 2.11 иллюстрирует первые три свойства. Четвертое свойство делает криптографическую хеш-функцию *криптографической* хеш-функцией, и чтобы описать его, необходимо немного углубиться в теорию. Есть несколько вариаций четвертого свойства, и все они желательны для криптографических хеш-функций (рис. 2.12).

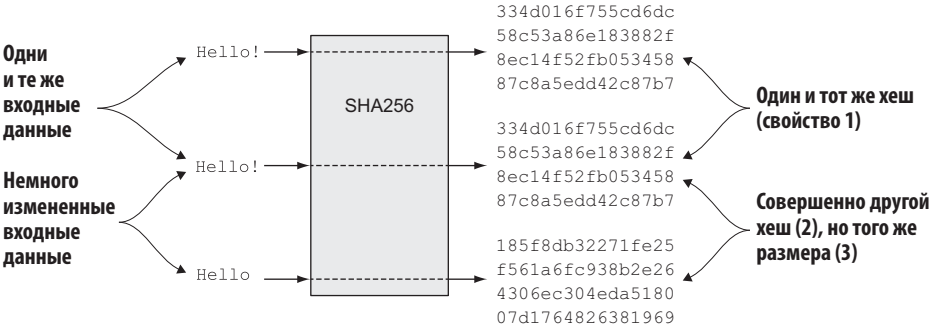


Рис. 2.11. Криптографическая хеш-функция SHA256 в действии. Для входной строки «Hello!» она всегда возвращает один и тот же хеш, но стоит немного изменить входную строку, например передать «Hello», и на выходе получится совершенно другой хеш

- *Устойчивость к коллизиям* — криптографическая функция должна усложнять поиск *разных* наборов входных данных, для которых *в результате получается одинаковый хеш*.
- *Устойчивость к определению исходного образа* — криптографическая функция должна усложнять *поиск другого исходного образа по известному хешу*.
- *Устойчивость к определению второго исходного образа* — криптографическая функция должна усложнять поиск *другого исходного образа с тем же хешем* по известному исходному образу и хеш-функции (а значит, и известному хешу).

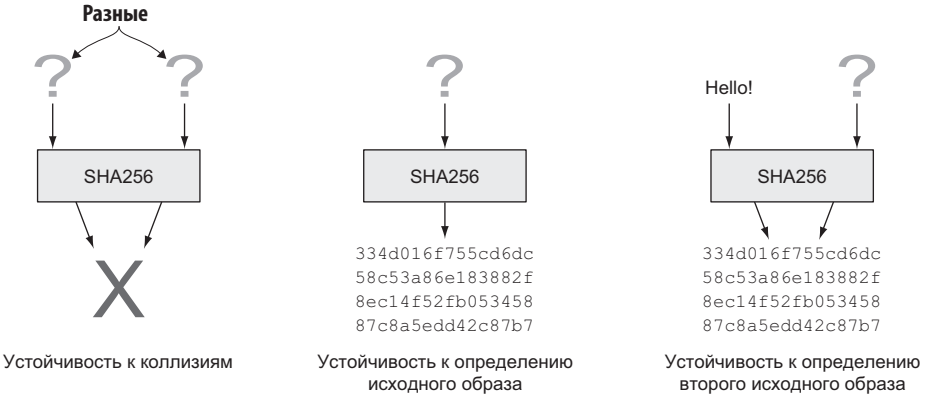
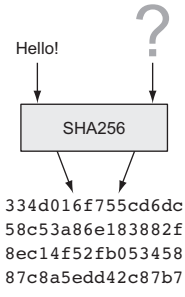


Рис. 2.12. Разные желаемые свойства криптографических хеш-функций. Для случая устойчивости к коллизиям под X подразумевается любой хеш, одинаковый для двух разных входных данных

Иллюстрация «усложнения»



Под словами «должна усложнять» в этом контексте подразумевается «усложнять до невозможности», то есть делать данную задачу вычислительно неразрешимой. Глупо даже пытаться проделать это. Далее, чтобы понять, что означает «усложнять», мы рассмотрим устойчивость к определению второго исходного образа, но аналогичный пример можно создать для любого из трех вариантов.

Допустим, мы решили найти входные данные, для которых SHA256 возвращает тот же хеш, что и для строки «Hello!»:

```
334d016f755cd6dc58c53a86e183882f8ec14f52fb05345887c8a5edd42c87b7
```

Мы не можем внести изменения в строку «Hello!» настолько небольшие, что функция «не заметит» их. Изменения *будут* замечены, и в результате получится совершенно другой хеш. Единственный способ найти строку, отличную от «Hello!», которая дает тот же хеш `334d016f ... d42c87b7`, — пробовать разные строки одну за другой и проверять получаемый хеш.

Попробуем (табл. 2.3).

Таблица 2.3. Найти строку, имеющую тот же хеш, что и строка «Hello!», почти невозможно

Входная строка	Хеш	Совпадает?
Hello1!	`82642dd9...2e366e64`	Нет
Hello2!	`493cb8b9...83ba14f8`	Нет
Hello3!	`90488e86...64530bae`	Нет
...	`...`	Нет, нет, ... нет
Hello9998!	`cf0bc6de...e6b0caa4`	Нет
Hello9999!	`df82680f...ef9bc235`	Нет
Hello10000!	`466a7662...ce77859c`	Нет
	`dee6a5d3...db479561`	Нет
Вся моя коллекция музыки	`a5bcb2d9...9c143f7a`	Нет

НАСКОЛЬКО ВЕЛИКО ЧИСЛО 2^{256}?

2^{256} примерно равно числу 10^{77}, сопоставимому с числом атомов во Вселенной. Поиск исходного образа по известному хешу SHA256 можно сравнить с попыткой перебрать все атомы во Вселенной в поисках нужного.

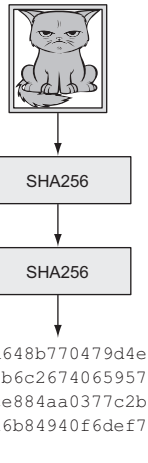
Как видите, наша попытка не увенчалась успехом. Посчитайте, сколько времени потребуется обычному компьютеру, чтобы найти такую строку. Допустим, он может вычислять около 60 миллионов хешей в секунду. В худшем случае ему придется выполнить 2^{255} попыток. В результате получается $2^{255} / (60 \times 10^6)$ с $\approx 10^{68}$ с $\approx 3 \times 10^{61}$ лет, или около 30 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 лет.

Полагаю, что попытки бессмысленны, я прав? Я не думаю, что покупка более мощного компьютера исправит ситуацию. Даже если у вас будет триллион компьютеров, выполняющих вычисления параллельно, вам понадобится около 3×10^{49} лет, чтобы перебрать все варианты.

Устойчивость к определению исходного образа, устойчивость к определению второго исходного образа и устойчивость к коллизиям чрезвычайно важны для Биткоин. Основная доля защищенности системы обусловлена этими свойствами.

ДВОЙНОЕ ХЕШИРОВАНИЕ SHA256

В Биткоин часто используется двойное хеширование SHA256:



Некоторые известные хеш-функции

В табл. 2.4 перечислено несколько разных криптографических хеш-функций. Некоторые из них уже считаются криптографически небезопасными.

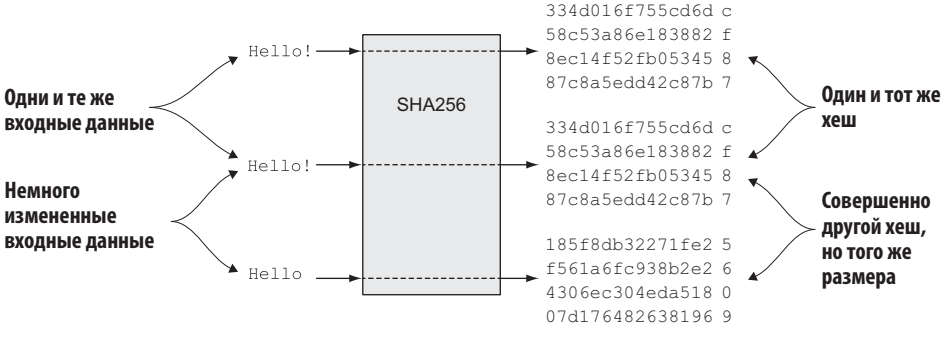
Как правило, когда в криптографической хеш-функции обнаруживается хотя бы одна коллизия, большинство криптографов начинают считать эту функцию небезопасной.

Таблица 2.4. Несколько криптографических хеш-функций. Некоторые из них считаются криптографически небезопасными

Функция	Размер хеша в битах	Безопасна до сих пор?	Используется в Биткоин?
SHA256	256	Да	Да
SHA512	512	Да	Да, в некоторых кошельках
RIPEMD160	160	Да	Да
SHA-1	160	Нет. Найдены коллизии	Нет
MD5	128	Нет. Найти коллизии не составляет труда. Алгоритм уязвим для атак поиска исходного образа, которые, впрочем, провести довольно затруднительно	Нет

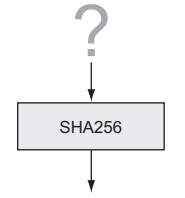
И снова о криптографических хешах

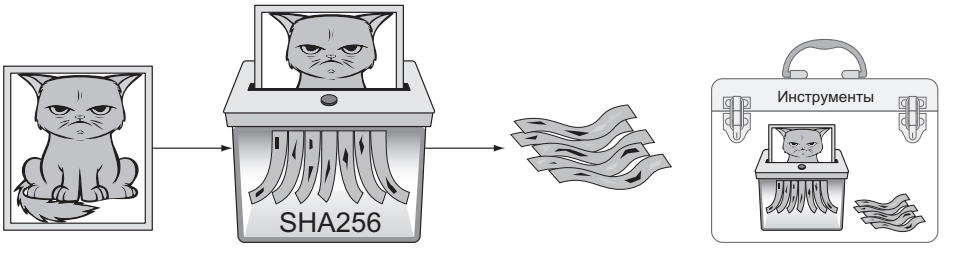
Криптографическая хеш-функция — это компьютерная программа, которая принимает некоторые данные и на их основе вычисляет большое число — криптографический хеш.



Практически невозможно найти входные данные, имеющие точно такой же хеш. Вот почему мы называем криптографические хеш-функции *однонаправленными*. Вам придется многократно опробовать разные варианты, чтобы найти нужный хеш.

В этой книге обсуждается множество важных тем. Познакомившись с конкретной темой, например с криптографическими хеш-функциями, вы можете добавить новый инструмент в свой ящик для инструментов. Ваш первый инструмент — криптографическая хеш-функция, которая изображена ниже как бумагорезательная машина; криптографический хеш можно сравнить с кучкой бумажных полосок, полученных в результате разрезания документа.





С этого момента мы будем использовать эти значки инструментов для представления криптографических хеш-функций и криптографических хешей, за некоторыми исключениями.

Упражнения

Для разминки

2.1. Хеш какого размера в битах возвращает SHA256?

2.2. Сколько байтов в хеше, возвращаемом SHA256?

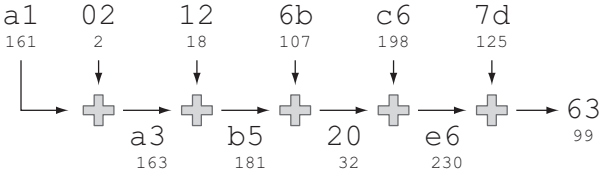
2.3. Что нужно, чтобы получить криптографический хеш для строки «hash me» (хешируй меня)?

2.4. Преобразуйте шестнадцатеричное число `061a` в двоичное и десятичное представления.

2.5. Получится ли у вас на практике изменить строку «cat» так, чтобы измененная строка имела тот же хеш, что и строка «cat»?

Придется пораскинуть мозгами

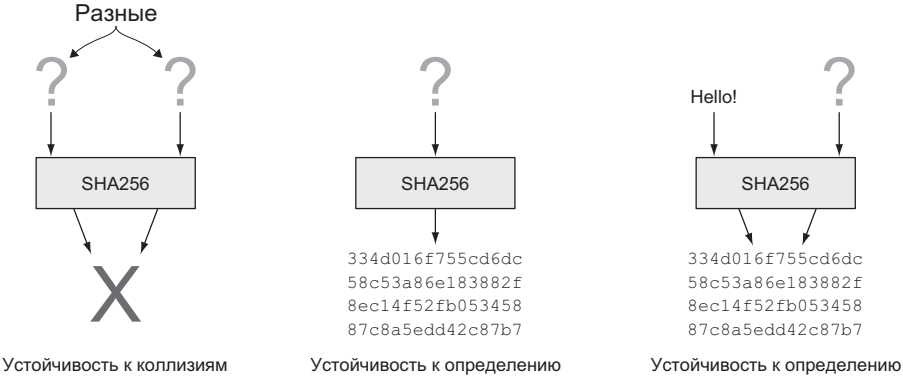
2.6. Упрощенная хеш-функция из раздела «Как работают криптографические хеш-функции?», повторно изображенная ниже, не является *криптографической* хеш-функцией. Какие два из четырех свойств криптографических хеш-функций в ней отсутствуют?



Напомню четыре свойства криптографических функций:

1. Для одних и тех же исходных данных генерируется один и тот же хеш.
2. Если входные данные изменить, на выходе получится другой хеш.
3. Хеш всегда имеет один и тот же фиксированный размер. Для SHA256 он равен 256 бит.
4. Метод полного перебора — единственный известный способ определить входные данные, дающие в результате заданный хеш.

2.7. Вернемся к примеру с изображением кота на жестком диске, когда мы записали криптографический хеш изображения на листке бумаги. Предположим, кто-то захотел изменить изображение кошки на вашем жестком диске без вашего ведома. Какой вариант четвертого свойства может помешать атакующему добиться успеха?



Цифровые подписи

В этом разделе мы посмотрим, как можно доказать кому-то, что вы подтверждаете платеж. Для этого используются *цифровые подписи*. Цифровая подпись — это цифровой эквивалент рукописной подписи. Разница в том, что рукописная подпись связана с человеком, а цифровая связана со случайным числом, которое называется *закрытым ключом*. Подделать цифровую подпись намного сложнее, чем рукописную.

Примеры использования цифровых подписей

Предположим, вы решили отправить фотографию своего любимого кота другу Фреду по электронной почте, но подозреваете, что изображение может быть злонамеренно или случайно повреждено во время передачи. Как бы вы с Фредом могли убедиться, что изображение, полученное Фредом, в точности соответствует отправленному вами?

Вы можете добавить в электронное письмо цифровую подпись для фотографии кота, а Фред может проверить эту цифровую подпись, чтобы удостовериться, что фотография является подлинной. Это делается в три этапа, как показано на рис. 2.13.

Шаг ❶ — *подготовка*. Вы создаете огромное случайное число — закрытый ключ — и используете его для создания цифровых подписей. Затем вы создаете *открытый ключ*, который используется для проверки подписей, созданных с помощью закрытого ключа. Открытый ключ *вычисляется* на основе закрытого ключа. Вы передаете открытый ключ Фреду лично, чтобы Фред был уверен, что он принадлежит вам.

Шаг ❷ — *подписывание*. Вы пишете письмо Фреду и вкладываете фотографию кота. Вы также используете свой закрытый ключ и фотографию, чтобы подписать ее. Результатом является цифровая подпись, которую также вкладываете в свое письмо. Затем вы отправляете письмо Фреду.

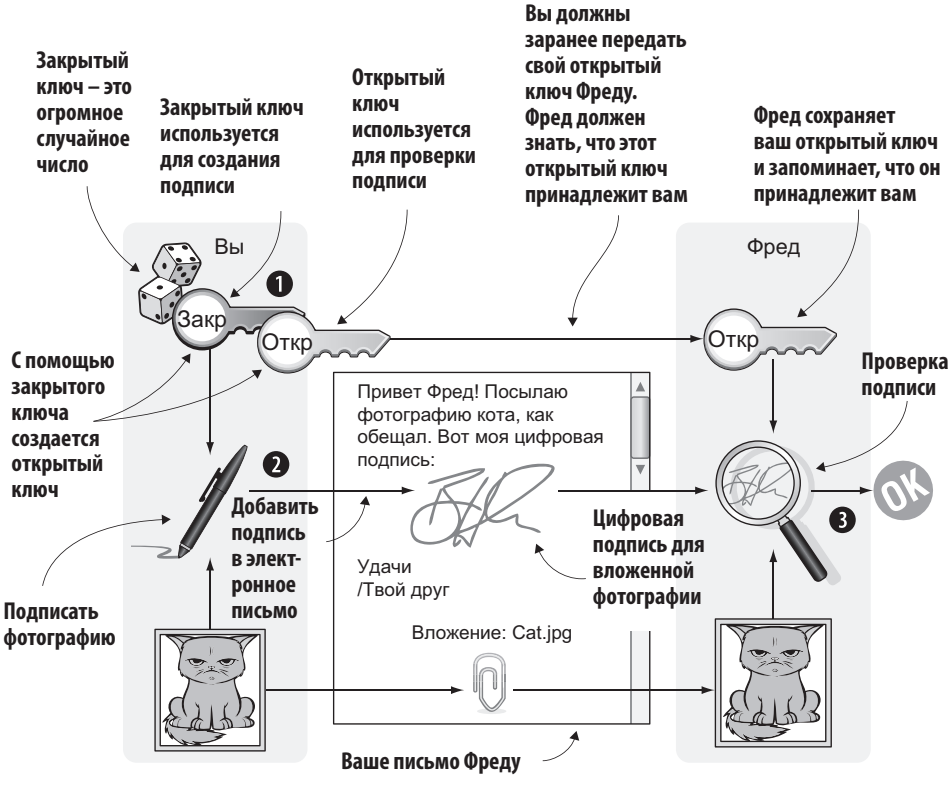


Рис. 2.13. Вы посылаете фотографию кота, подписанную цифровой подписью. Фред проверяет подпись, чтобы убедиться, что он получил именно ту фотографию, которую вы подписали

Шаг ❸ — *проверка*. Фред получает ваше электронное письмо, но он обеспокоен, что фотография могла быть повреждена, поэтому он хочет проверить подпись. Он использует открытый ключ, полученный от вас на шаге ❶, цифровую подпись в электронном письме и фотографию. Если подпись или фотография изменились с момента создания подписи, проверка не увенчается успехом.

Улучшение безопасности жетонов на булочки

Пришло время вернуться к нашей таблице с жетонами на булочки. Компания растет, и Лизе трудно запомнить всех. Она заметила, что некоторые люди нечестны. Например, Мэллори говорит, что она Энни, чтобы обманом заставить Лизу передать в кафе жетоны, принадлежащие Энни, а не Мэллори. В результате Лиза решила заставить всех подписывать свои просьбы цифровыми подписями, вкладывая их в свои электронные письма, как показано на рис. 2.14.

ПАРА КЛЮЧЕЙ ИСПОЛЬЗУЕТСЯ МНОГОКРАТНО

Пара ключей создается только один раз. Один и тот же закрытый ключ можно использовать много раз для создания цифровой подписи.

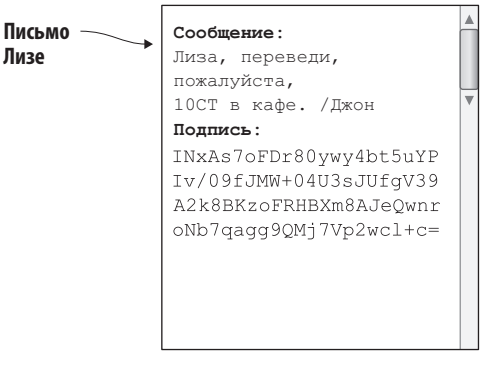


Рис. 2.14. Джон должен подписать свою просьбу выполнить платеж, вложив цифровую подпись в письмо

Предположим, Джон — новый сотрудник. Компания дала ему несколько жетонов на булочки в качестве приветственного подарка. Теперь Джон хочет купить булочки в кафе на 10 СТ. Он должен поставить цифровую подпись под просьбой передать жетоны в кафе. На рис. 2.15 показано, как это делается.

По аналогии с письмом Фреду в предыдущем разделе, этот процесс выполняется в три этапа (сравните с шагами на рис. 2.13, чтобы увидеть сходство):

❶ Джон выполняет подготовительные операции, генерируя пару ключей, сохраняет свой закрытый ключ в безопасности и вручает открытый ключ Лизе. Все это выполняется как один шаг.

❷ Джон хочет купить булочки. Он пишет письмо и подписывает его своим закрытым ключом. Затем он посылает письмо и цифровую подпись Лизе.

❸ Лиза проверяет подпись в сообщении, используя открытый ключ Джона, и вносит изменения в электронную таблицу.

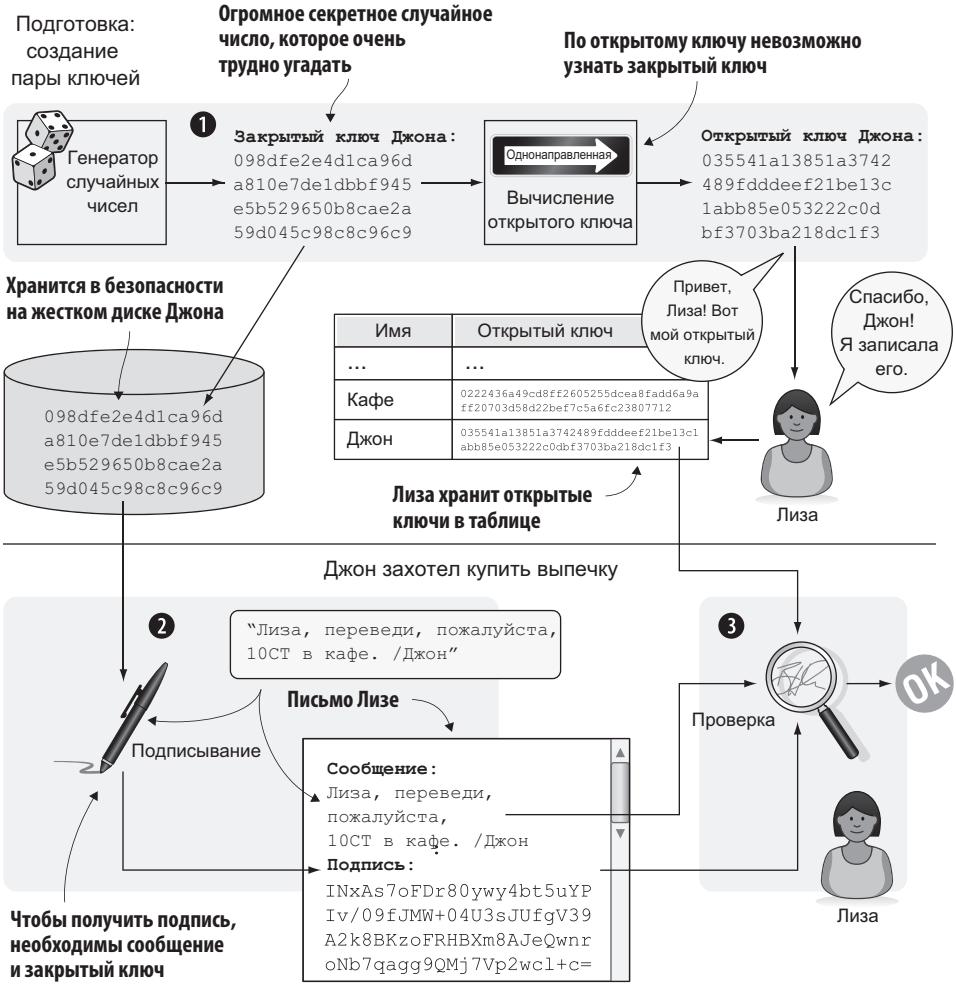
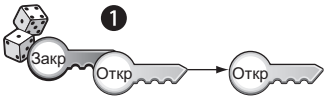


Рис. 2.15. Процесс добавления цифровой подписи.
❶ Джон создает пару ключей и передает открытый ключ Лизе.
❷ Джон подписывает сообщение закрытым ключом.
❸ Лиза проверяет подпись открытым ключом, принадлежащим Джону

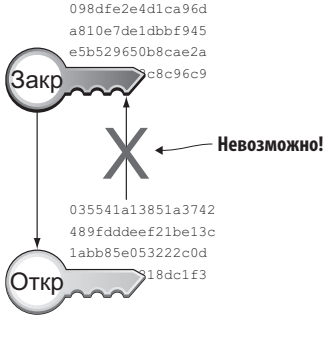
Подготовка: создание пары ключей



Процессы подписания и проверки основаны на паре ключей. Джону нужен закрытый ключ, чтобы подписать платеж, а Лизе нужен открытый ключ Джона, чтобы проверить подпись Джона. Джон должен подготовиться к этому, создав пару ключей. Для этого он сначала генерирует закрытый ключ, а затем, на основе закрытого ключа, вычисляет открытый ключ, как показано на рис. 2.16.

Рис. 2.16. Джон создает пару ключей. Закрытый ключ — это огромное случайное число, а открытый ключ вычисляется на основе закрытого ключа. Джон сохраняет свой закрытый ключ у себя на жестком диске, а открытый ключ передает Лизе

Чтобы сгенерировать огромное 256-битное случайное число, Джон использует генератор случайных чисел. Генератор случайных чисел имеется практически во всех операционных системах. С этого момента случайное число является закрытым ключом Джона. Затем, с использованием функции создания открытого ключа, на основе закрытого ключа вычисляется открытый ключ.



Вычисление открытого ключа является однонаправленной функцией, как и криптографические хеш-функции; из открытого ключа нельзя получить закрытый ключ. Безопасность цифровых подписей в значительной степени зависит от этой функции. Кроме того, любые попытки вычислить открытый ключ всегда будут давать в результате один и тот же открытый ключ.

Открытый ключ имеет длину 33 байта (66 шестнадцатеричных цифр). Он длиннее закрытого ключа, длина которого составляет 32 байта (64 шестнадцатеричных числа). Наличие «лишнего» байта обусловлено особенностью работы функции вычисления открытого ключа. Это довольно сложная тема, и мы обсудим ее в главе 4. К счастью, не нужно быть экспертом в криптографии, чтобы понять, как работают подписи с точки зрения пользователя.

Два способа использования пары ключей

Ключи можно использовать для шифрования и расшифровывания данных. Цель шифрования — сделать сообщение нечитаемыми для всех, кроме имеющих соответствующий ключ для расшифровывания.

Закрытый и открытый ключи можно считать парой, потому что они тесно связаны: открытый ключ можно использовать для шифрования сообщений, которые можно расшифровать только с помощью закрытого ключа, а закрытым ключом можно шифровать сообщения, которые расшифровываются только открытым ключом (рис. 2.17).

ПРИМЕЧАНИЕ

Процесс создания цифровой подписи изображен рис. 2.17 справа. Левую часть мы не будем использовать в этой книге.

Как показано на рис. 2.17 слева, только Джон сможет прочитать зашифрованное сообщение, потому что он — единственный, кто имеет доступ к своему закрытому ключу. Система Биткоин вообще не использует эту функцию открытых и закрытых ключей. Она используется, только когда две стороны хотят пообщаться в закрытом режиме, как, например, когда вы выполняете операции в онлайн-банке. Когда вы видите в адресной строке веб-браузера значок с маленьким изображением замка, это означает, что для защиты ваших действий используется процесс, изображенный слева на рис. 2.17.

Как показано на рис. 2.17 справа, Лиза может расшифровать сообщение, потому что у нее есть открытый ключ, принадлежащий Джону. Эта возмож-

ность используется для добавления цифровой подписи. Было бы неправильно использовать закрытый ключ для шифрования секретных сообщений, потому что открытый ключ и есть открытый. Любой сможет расшифровать сообщение, использовав открытый ключ. Цифровые подписи, напротив, не являются секретными сообщениями. Вскоре мы еще вернемся к цифровым подписям. Но сначала повторим пройденное.

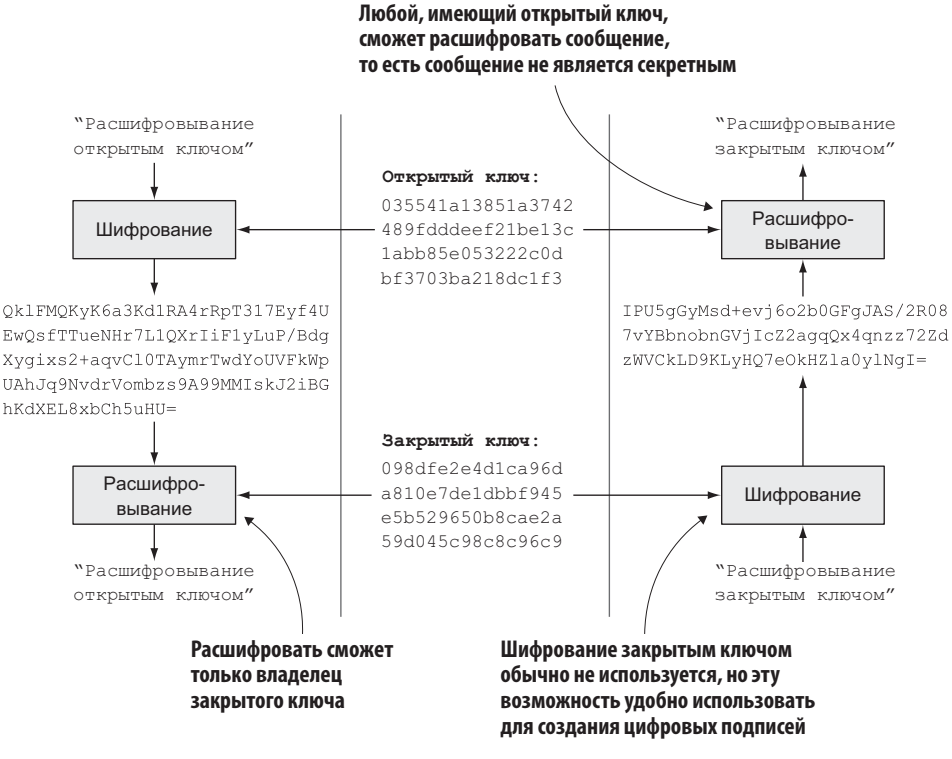
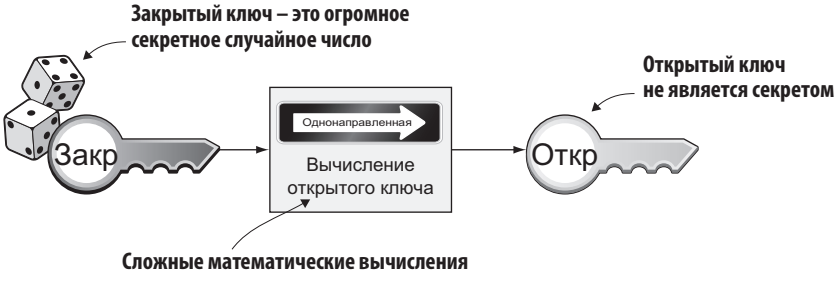


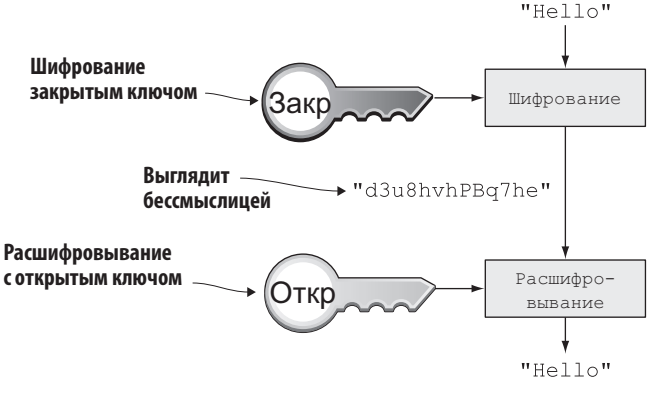
Рис. 2.17. Шифрование и расшифровывание с помощью открытого и закрытого ключей. Слева: шифрование открытым и расшифровывание открытым ключом. Справа: шифрование закрытым и расшифровывание закрытым ключом

Еще раз о парах ключей

Перечислим, что мы узнали об открытых и закрытых ключах. При создании пары ключей сначала создается закрытый ключ. Закрытый ключ — это секретное огромное случайное число. На основе закрытого ключа вычисляется открытый ключ.



Закрытый ключ можно использовать для шифрования сообщения, которое можно расшифровать только с помощью открытого ключа.



Шифрование и расшифровывание на рисунке выше — это суть цифровой подписи. Этот процесс *не* подходит для передачи секретных сообщений, потому что открытый ключ обычно широко известен.

Обратный процесс, когда открытый ключ используется для шифрования, а закрытый ключ — для расшифровывания, тоже имеет широкое применение. Он используется для отправки секретных сообщений. Биткоин не использует его.

На чем мы остановились?

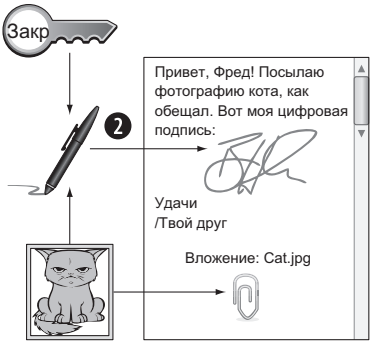
Цифровые подписи кратко упоминались в главе 1, где Алиса подписала свою транзакцию на перевод 1 BTC Бобу, использовав свой закрытый ключ (рис. 2.18).

Рис. 2.18. Цифровые подписи в Биткоин

Джон создал пару ключей и собирается подписать платеж в кафе своим закрытым ключом, чтобы Лиза могла убедиться, что платеж выполняет действительно Джон. Лиза проверяет подпись, используя открытый ключ Джона.

Джон подписывает свой платеж

Рассмотрим подробнее, как в действительности происходит подписывание (рис. 2.19).



Джон хочет подписать сообщение: «Лиза, переведи, пожалуйста, 10 CT в кафе. /Джон». Функция, генерирующая подпись, хеширует это сообщение с помощью алгоритма SHA256, и на выходе получается 256-битное число. Затем этот хеш-код шифруется закрытым ключом Джона. В результате получается цифровая подпись, которая выглядит так:

```
INxAs7oFDr80ywy4bt5uYPIv/09fJMW+04U3sJUfgV39
A2k8BKzoFRHBXm8AJeQwnroNb7qagg9QMj7Vp2wcl+c=
```

Подпись — это зашифрованный хеш сообщения. Если Джон подпишет другим закрытым ключом или сообщение немного изменится на пути к Лизе, подпись будет выглядеть совершенно иначе.

ПОДПИСИ В БИТКОИН

В настоящее время подобные подписи используются в Биткоин для большинства платежей, но это не единственный способ аутентификации платежей.

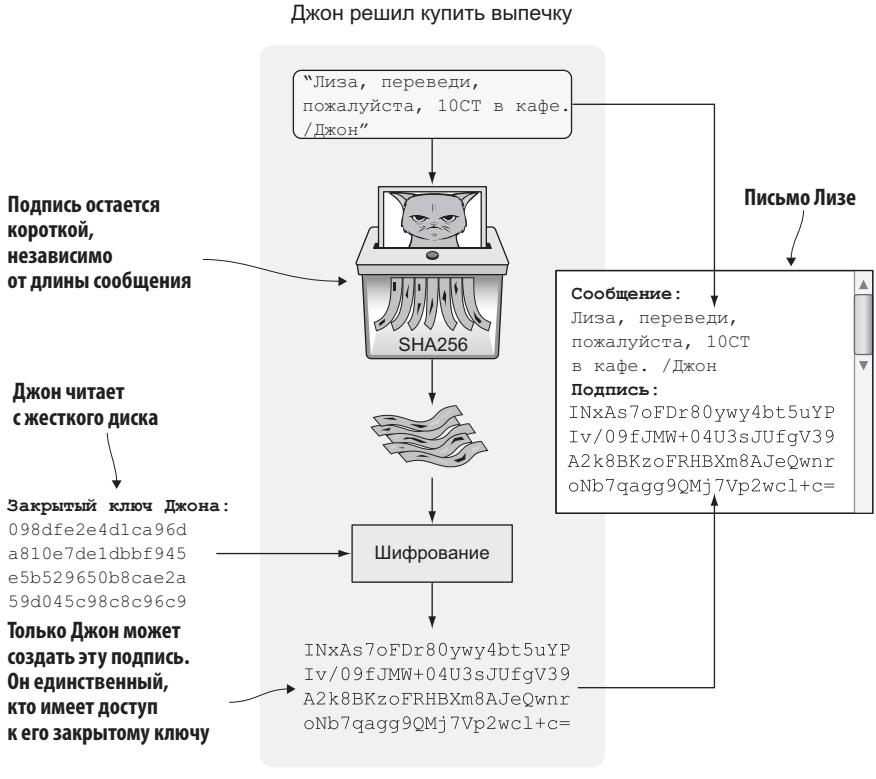


Рис. 2.19. Джон подписывает просьбу о переводе 10 CT в кафе цифровой подписью. Сообщение Лизе сначала хешируется и затем шифруется закрытым ключом Джона. Письмо Лизе содержит обычный текст сообщения и подпись

Например, для сообщения «Лиза, переведи, пожалуйста, 10 CT Мэллори. / Джон» будет сгенерирована другая подпись:

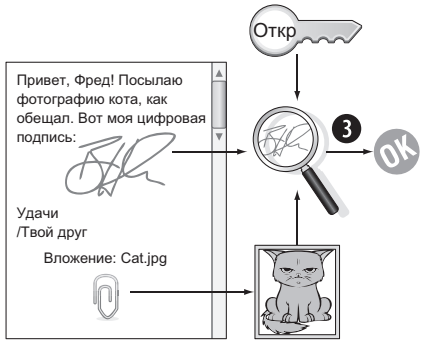
```
ILDtL+AVMmOrcrvCRwnsJUJUtzedNkSoLb7OLRoH2iaD
G1f2WX1dAOTYkszR1z0TfTVIVwdAlD0W7B2hBTAzFkk=
```

Она не похожа на предыдущую подпись. Это хорошо для Джона, потому что он знает, что его подпись нельзя использовать для подделки сообщений.

Итак, Джон написал письмо Лизе. Это электронное письмо содержит текст сообщения и его подпись. После этого Джон отправляет электронное письмо Лизе.

Лиза проверяет подпись

Лиза открывает письмо и видит, что оно подписано человеком, который утверждает, что он — Джон, поэтому она открывает свою таблицу и извлекает из нее открытый ключ Джона (рис. 2.20).



Действия, изображенные на рис. 2.20, Лиза выполняет, чтобы убедиться, что просьба о переводе жетонов подписана закрытым ключом человека, написавшего сообщение. В сообщении говорится, что оно написано Джоном. На днях она получила открытый ключ Джона и сохранила его в своей таблице открытых ключей. Вот что она имеет на данный момент:

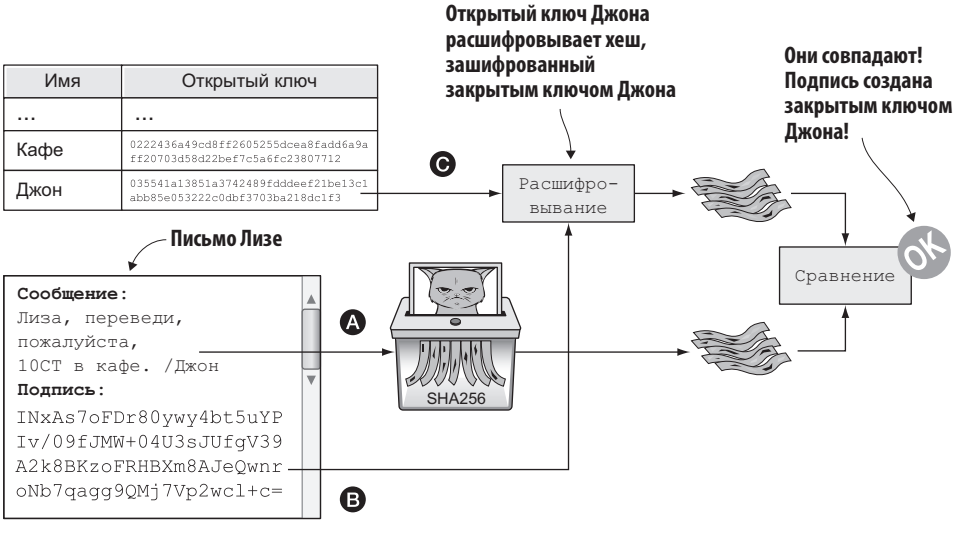


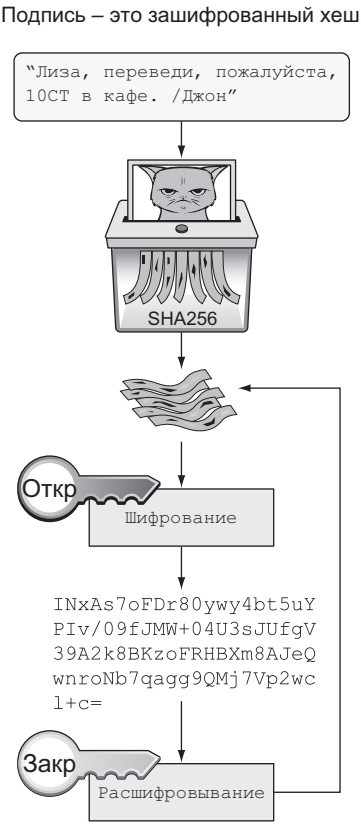
Рис. 2.20. Лиза использует сообщение Ⓐ, цифровую подпись Ⓑ и открытый ключ Джона Ⓒ, чтобы убедиться, что это сообщение подписано закрытым ключом Джона

Ⓐ Сообщение: «Лиза, переведи, пожалуйста, 10 CT в кафе. /Джон».

Ⓑ Подпись `INxAs7oFDr8`...

Ⓒ Открытый ключ Джона, который она только что нашла в своей таблице.

Джон зашифровал хеш сообщения своим *закрытым* ключом. Этот зашифрованный хеш служит подписью. Когда Лиза расшифрует подпись Ⓑ *открытым* ключом Ⓒ Джона, она должна получить хеш, совпадающий с хешем сообщения Ⓐ в письме.



Лиза расшифровывает подпись Ⓑ с помощью открытого ключа Ⓒ из своей таблицы открытых ключей и получает в результате большое число. Если число совпадает с хешем сообщения Ⓐ, это доказывает, что для подписи сообщения использовался закрытый ключ Джона. Чтобы получить хеш, Лиза берет текст сообщения Ⓐ из письма в точности как написано и хеширует его так же, как это сделал Джон, когда создавал подпись. Этот хеш затем сравнивается с расшифрованной подписью. Если хеш сообщения и дешифрованная подпись совпадают, значит, подпись действительна.

Обратите внимание, что подобный способ проверки возможен, только если Джон и Лиза используют одну и ту же схему создания цифровых подписей. Она должна быть согласована заранее, но чаще используется стандартная процедура. В Биткоин все точно знают, какую схему создания цифровых подписей использовать.

Теперь Лиза может быть уверена, что никто не пытается ее одурачить. Она обновляет электронную таблицу, добавляя перевод Джону, как показано на рис. 2.21.

От кого	Кому	Количество CT
...	...	...
Компания	Джон	100
Джон	Кафе	10

Новая запись! Джон подписал просьбу о переводе, и Лиза убедилась в достоверности подписи. Она проверила баланс Джона и добавила запись о переводе в электронную таблицу

Рис. 2.21. После проверки подписи сообщения Лиза добавляет запись о переводе жетонов Джону

Безопасность закрытого ключа

Джон полностью контролирует свои жетоны на булочки, потому что владеет закрытым ключом. Никто, кроме Джона, не сможет использовать его жетоны, потому что он единственный, кто имеет доступ к своему закрытому ключу. Если его закрытый ключ украдут, он может потерять все свои жетоны.

На следующий день, придя утром на работу, Джон берет свой ноутбук со стола и идет прямо в кафе, чтобы купить две утренние булочки. Он открывает свой ноутбук и пишет письмо Лизе:

> Доброе утро, Лиза! Переведи, пожалуйста, 20 CT в кафе. /Джон
>
> Подпись:
>
> ```
> H1CdE34cRuJDsHo5VnpvKqllC5JrMJ1jWcUjL2VjPbsj
> X6pi/up07q/gWxStb1biGU2fjcKpT4DIxlNd2da9x0o=
> ```

Он посылает письмо с сообщением и подписью Лизе. Но кафе не продает ему булочки. Сотрудник кафе говорит, что пока не получил платеж на 20 CT. Обычно Лиза быстро проверяет и выполняет переводы.

Джон открывает электронную таблицу — если вы помните, она доступна ему только для чтения — и выполняет поиск записей по имени «Джон». На рис. 2.22 показано, что он видит.

От кого	Кому	Количество CT
…	…	…
Компания	Джон	100
Джон	Кафе	10
Джон	Мелисса	90

Общий баланс 100 CT

Это вчерашний перевод
Баланс 90 CT

Что? Джон не переводил
Мелиссе 90 CT!

Рис. 2.22. Кто-то украл жетоны у Джона. Кто такая Мелисса и как такое возможно? Джон не подписывал никаких таких переводов

Джон отправляется в кабинет к Лизе, надеясь получить объяснение. Она отвечает, что получила сообщение, подписанное закрытым ключом Джона, с просьбой перевести жетоны новой коллеге Мелиссе. Лиза даже показала ему сообщение и подпись. Вчера в компанию поступило на работу несколько новых сотрудников, но среди них нет Мелиссы. Лизе больше не нужны имена, только открытые ключи и подписи. Имена ей нужны, только чтобы найти правильный открытый ключ в таблице.

Случившееся объясняется тем, что Мэллори:

1. Скопировала закрытый ключ Джона. Ноутбук Джона оставался на его рабочем столе всю ночь. Любой мог извлечь жесткий диск из ноутбука и отыскать на нем закрытый ключ.

2. Создала новую пару ключей и послала новый открытый ключ Лизе со следующим текстом:

   Привет, Лиза. Меня зовут Мелисса, и я новый сотрудник.

   Мой открытый ключ

   ```
   02c5d2dd24ad71f89bfd99b9c2132f796fa746596a06f5
   a33c53c9d762e37d9008
   ```

3. Послала Лизе мошенническое письмо, подписанное украденным закрытым ключом Джона:

   Привет, Лиза. Переведи, пожалуйста, 90 CT Мелиссе. Спасибо, Джон

   Подпись:

   ```
   IPSq8z0IyCVZNZNMIgrOz5CNRRtRO+A8Tc3j9og4pWbA
   H/zT22dQEhSaFSwOXNp0lOyE34d1+4e30R86qzEbJIw=
   ```

Лиза проверила перевод, сделанный на шаге 3, сочла его действительным и выполнила. Джон просит Лизу отменить этот мошеннический, по его мнению, перевод. Но Лиза отказывается это сделать. Она считает, что перевод является действительным. Если Джон позволил кому-то увидеть его закрытый ключ, то это его проблема, а не Лизы. Именно поэтому она пользуется большим доверием у сотрудников компании — она всегда выполняет свои обещания.

Джон создает новую пару ключей и просит Лизу добавить новый открытый ключ под именем Джон2. Как Джону защитить свой новый закрытый ключ и вместе с тем иметь возможность быстро получить доступ к нему, когда ему захочется купить булочку? Джон почти уверен, что на этом ключе у него не будет более 1000 жетонов.

БЕЗОПАСНОСТЬ — ЭТО ВАША ОТВЕТСТВЕННОСТЬ

Вся ответственность за безопасность ваших закрытых ключей лежит только на вас.

Безопасность электронной таблицы теперь основана не на личном знакомстве с Лизой, а на наличии у нее открытых ключей всех сотрудников. В некотором смысле безопасность ухудшилась, потому что украсть закрытый ключ

Джона может быть проще, чем заставить Лизу думать, что Мэллори — это Джон. Теперь все зависит от того, как Джон защитит свой закрытый ключ. Важно отметить, что безопасность закрытого ключа Джона полностью зависит от него самого. Никто не сможет восстановить закрытый ключ Джона, если он его потеряет. И Лиза твердо уверена, что не должна отменять «мошеннические» переводы только потому, что Джон небрежно относится к безопасности.

Если Джон будет хранить свой закрытый ключ в открытом виде в общей папке во внутренней сети компании, любой сможет скопировать его и использовать для кражи жетонов на булочки. Но если Джон сохранит закрытый ключ на жестком диске своего ноутбука в зашифрованном файле, защищенном надежным паролем, получить копию его ключа будет гораздо сложнее. Для этого злоумышленник должен:

- иметь доступ к жесткому диску Джона;
- знать пароль Джона.

Если на закрытом ключе Джона никогда не будет больше 50 СТ, он может особенно не волноваться о безопасности. Но кафе, которому поступает что-то около 10 000 СТ в день, безопасность очень волнует. Джону и кафе, вероятно, нужны разные стратегии хранения закрытых ключей.

Выбирая между безопасностью и удобством, приходится чем-то жертвовать. Например, вы можете хранить закрытый ключ в зашифрованном виде на ноутбуке, лежащем в банковском сейфе. Если вы захотите купить булочки, вам придется пойти в банк, вынуть ноутбук из сейфа, расшифровать секретный ключ с помощью пароля, написать и подписать письмо Лизе и сохранить это письмо на флешку. Затем вам нужно положить ноутбук обратно в сейф, принести флешку в офис и отправить электронное письмо Лизе. Закрытый ключ никогда не покидает пределы ноутбука в сейфе. Очень безопасно и очень неудобно.

С другой стороны, вы можете хранить закрытый ключ в открытом виде на своем мобильном телефоне. Ключ всегда будет у вас под рукой, и вы сможете подписать сообщение в течение нескольких секунд, как только у вас появится желание использовать жетоны. Очень небезопасно и очень удобно.

Вот некоторые достоинства и недостатки вариантов, изображенных на рис. 2.23:

- *Доступен и недоступен из сети* — под доступностью из сети подразумевается хранение закрытого ключа на устройстве с доступом к сети,

например на мобильном телефоне или ноутбуке общего назначения. Под недоступностью из сети подразумевается хранение закрытого ключа на листе бумаги или на компьютере без доступа к сети. Хранить ключ на устройстве, доступном из сети, опасно, потому что вредоносные программы на вашем компьютере, такие как компьютерные вирусы, могут тайком отправить ваш закрытый ключ кому-то. Если устройство недоступно из сети, никто не сможет скопировать закрытый ключ, не имея физического доступа к устройству.

Рис. 2.23. Защищенность от злоумышленников. Обратите внимание, что более безопасные варианты также наименее удобны

- *В открытом и в зашифрованном виде* — если закрытый ключ хранится в открытом виде в простом текстовом файле на жестком диске вашего компьютера, любой, имеющий доступ к вашему компьютеру, сможет удаленно через компьютерную сеть или физически скопировать закрытый ключ. Опасность также представляют любые вирусы, жертвой которых может стать ваш компьютер. Многих из этих опасностей можно избежать, зашифровав закрытый ключ с помощью пароля, известного только вам. В этом случае, чтобы получить закрытый ключ, злоумышленнику потребуется получить доступ к вашему жесткому диску и узнать секретный пароль.
- *Ключ целиком и разбит на фрагменты* — люди обычно хранят свои закрытые ключи целиком на одном компьютере. Это удобно — чтобы тратить жетоны, нужен только один компьютер. Чтобы украсть такой закрытый ключ, злоумышленник должен получить доступ к вашему жесткому диску. Но если закрытый ключ разделить на три части (есть хорошие и плохие способы сделать это — будьте осторожны) и сохранить их на трех разных компьютерах, то злоумышленнику придется получить доступ к жестким дискам трех компьютеров. Это гораздо сложнее, потому что он должны знать, какие три компьютера атаковать, а также успешно атаковать их. Осуществление платежей в этом случае — весьма хлопотное дело, но очень безопасное.

Вы можете использовать любую комбинацию этих методов для хранения своих ключей. Но, как правило, чем выше защищенность от злоумышленников, тем выше риск, что вы случайно потеряете доступ к своему ключу. Например, если вы храните закрытый ключ в зашифрованном виде на своем жестком диске, вы рискуете потерять свой ключ из-за сбоя компьютера или позабыв пароль. В этом смысле чем надежнее вы храните свой ключ, тем менее надежно он защищен.

Повторение

Лиза решила проблему с людьми, пытающимися провести платеж от чужого имени. Теперь она требует от всех плательщиков добавлять цифровую подпись в письмо с просьбой о переводе жетонов. Каждому пользователю этой системы нужен закрытый и открытый ключ. Лиза следит за тем, кому и какой открытый ключ принадлежит. Отныне электронное письмо с просьбой о переводе должно быть подписано цифровой подписью, зашифрованной закрытым ключом просящего. Благодаря такому подходу Лиза сможет проверить подпись и убедиться, что ее не одурачили. Суть в том, что пока Джон хранит свой закрытый ключ при себе, никто не сможет тратить его жетоны.

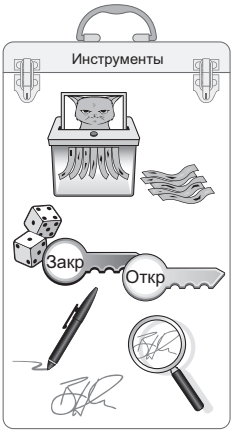
Добавим «письмо Лизе» в нашу таблицу понятий (табл. 2.5).

Таблица 2.5. Добавление «письма Лизе» в перечень ключевых понятий

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
Электронная таблица	Блокчейн	Глава 6
Письмо Лизе	**Транзакция**	**Глава 5**
Запись в электронной таблице	Транзакция	Глава 5
Лиза	Майнер	Глава 7

Таблица 2.6. Примечания к релизу, жетоны на булочки 2.0

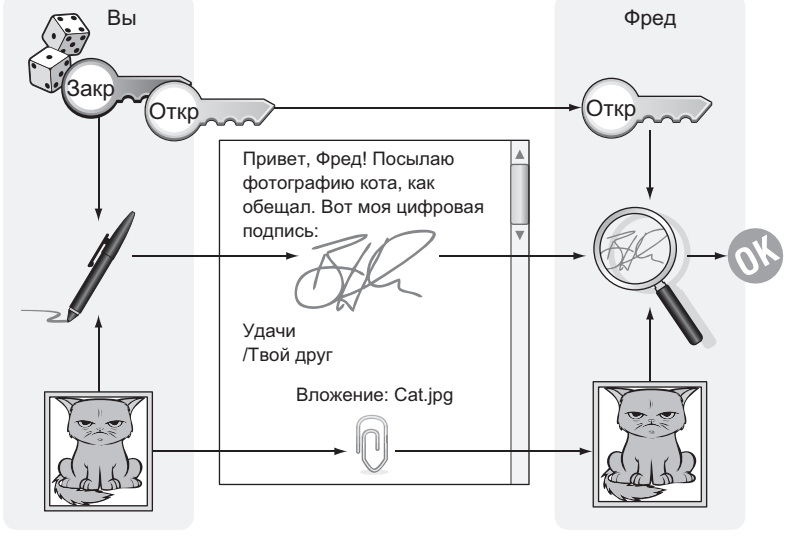
Версия	Особенность	Как реализована
НОВАЯ 2.0	Защищенная платежная система	Цифровые подписи решают проблему самозванцев
1.0	Простая платежная система	Основывается на доверии Лизе и личных знакомствах
	Конечный объем денег	Лиза получает в награду 7200 CT ежедневно; величина вознаграждения уменьшается вдвое каждые четыре года



В главе 5 мы заменим понятие «письмо Лизе» понятием «транзакция». Транзакция заменит не только понятие письма Лизе, но также понятие записи в электронной таблице. Настало время выпустить версию 2.0 системы жетонов на булочки (табл. 2.6).

Все сотрудники по-прежнему доверяют Лизе, что она никогда и ни за что не изменит электронную таблицу, кроме как для фиксации подписанных переводов жетонов. При желании Лиза могла бы украсть чьи-либо жетоны, просто добавив перевод в электронную таблицу. Но она ни за что не сделает это... или сделает?

Теперь у вас есть много новых инструментов в вашем ящике: пары ключей, цифровая подпись, подписывание и проверка.



Упражнения

Для разминки

2.8. В настоящее время Лиза получает за свою работу 7200 CT в день. Почему количество жетонов не будет увеличиваться до бесконечности? Почему через 10 000 дней в системе не будет 7200 × 10 000 = 72 миллиона CT?

2.9. Как другие сотрудники узнают, что Лиза приписывает себе лишние жетоны?

2.10. Как создается закрытый ключ?

2.11. Какой ключ используется для подписывания сообщения цифровой подписью?

2.12. Процесс подписывания основан на хешировании сообщения. Почему?

2.13. Что должна сделать Мэллори, чтобы украсть жетоны Джона?

Придется пораскинуть мозгами

2.14. Предположим, у вас есть закрытый ключ, и вы дали свой открытый ключ другу Фреду. Расскажите, как Фред сможет отправить вам секретное сообщение, прочитать которое сможете только вы.

2.15. Предположим, вы (допустим, что вас зовут Лора) и Фред владеете ключами из предыдущего упражнения. Теперь вы хотите отправить сообщение Фреду:

> Привет, Фред! Мы сможем встретиться завтра в Тиффани на закате?
> /Лора

Расскажите, как бы вы подписали это сообщение, чтобы Фред был уверен, что оно действительно написано вами. Опишите шаги, которые должен предпринять Фред.

Итоги

- Биткоины создаются как вознаграждение узлам, защищающим блокчейн.
- Вознаграждение уменьшается наполовину каждые четыре года, чтобы ограничить объем денежной массы.
- Для обнаружения изменений в файле или в сообщении можно использовать криптографические хеш-функции.
- Воссоздать исходный образ по криптографическому хешу невозможно. Исходный образ — это входные данные, результат хеширования которых предопределен.

- Цифровые подписи позволяют подтвердить подлинность платежа. Только законный владелец биткоинов может их потратить.
- Проверяющему цифровую подпись не требуется знать, *кто* ее создал. Ему достаточно знать, что подпись создана закрытым ключом, принадлежащим заявителю.
- Чтобы получать биткоины или жетоны на булочки, вам нужен открытый ключ. Сначала вы должны создать закрытый ключ, который будете хранить втайне от других, а затем вычислить открытый ключ на основе закрытого ключа.
- Есть несколько стратегий хранения закрытых ключей: от самых простых и небезопасных, когда ключ хранится в открытом виде в мобильном телефоне, до сложных и высокозащищенных, когда ключ делится на части, шифруется и хранится на нескольких автономных устройствах.
- Как правило, чем более защищен секретный ключ от кражи, тем выше риск случайно потерять его, и наоборот.

3 Адреса

Эта глава охватывает следующие темы:

- ✓ основы конфиденциальности;
- ✓ замена имен хешами открытых ключей;
- ✓ защита от дорогостоящих опечаток.

К концу этой главы в электронной таблице жетонов на булочки уже не будет личных имен — мы заменим их хешами открытых ключей. Это полезно с точки зрения конфиденциальности. Никто не сможет легко понять, кто кому платит, и другим будет труднее извлекать информацию из электронной таблицы и видеть, сколько жетонов потратил кто-то из ваших коллег. Лиза тоже считает это полезным, потому что ей не придется вести таблицу имен и открытых ключей.

При переходе на использование хешей открытых ключей в электронной таблице сотрудники компании больше не будут указывать свои имена в письмах Лизе. Вместо имен они будут использовать строки шестнадцатеричного кода, представляющие хеши открытых ключей. Но это означает, что возрастает риск допустить опечатку. Если вы допустите опечатку, ваши жетоны могут сгореть!

Кто-то из сотрудников изобрел адреса жетонов (адреса Биткоин), защищающие от потери денег из-за ошибок при вводе (рис. 3.1). Адреса жетонов

используются пользователями для переводов друг другу, почти как адреса электронной почты, но они отсутствуют в электронной таблице.

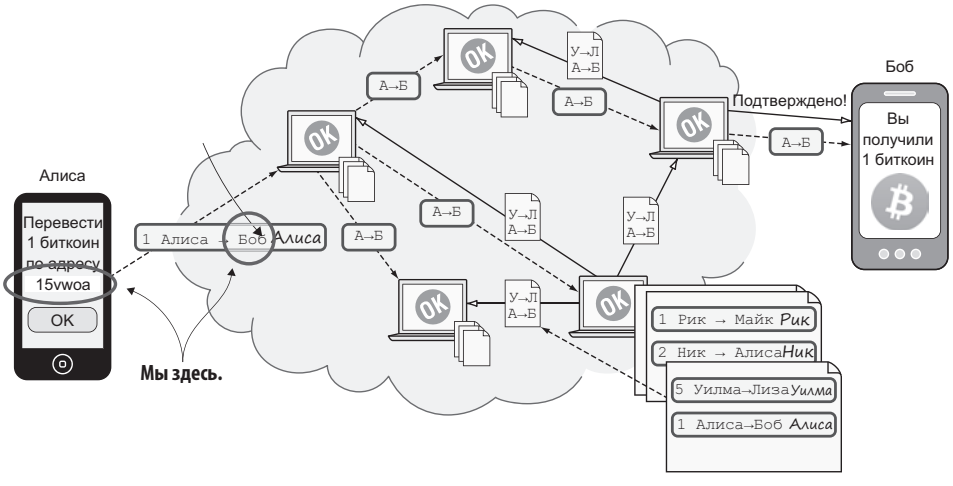


Рис. 3.1. Адреса жетонов на булочки в точности повторяют адреса Биткоин. В основном они используются программным кошельком

Раскрыты привычки потребления булочек

У вас и многих ваших коллег есть медицинская страховка от страховой компании Acme Insurances. Компания Acme убедила Джона передать им копию электронной таблицы. Сотрудники Acme заметили, что могут скорректировать страховую премию или привычки работников потреблять булочки (рис. 3.2) в случае возможного страхового спора.

ACME INSURANCES

Эта очень неэтичная страховая компания предпринимает серьезные попытки шпионить за вашими привычками, чтобы «скорректировать» страховую премию.

Другой неприятный факт, связанный с электронной таблицей, заключается в том, что любой сотрудник сможет легко увидеть баланс других сотрудников, а также узнать их привычки в потреблении булочек.

Сотрудники попросили Лизу найти решение этих проблем, иначе они перестанут использовать электронную таблицу.

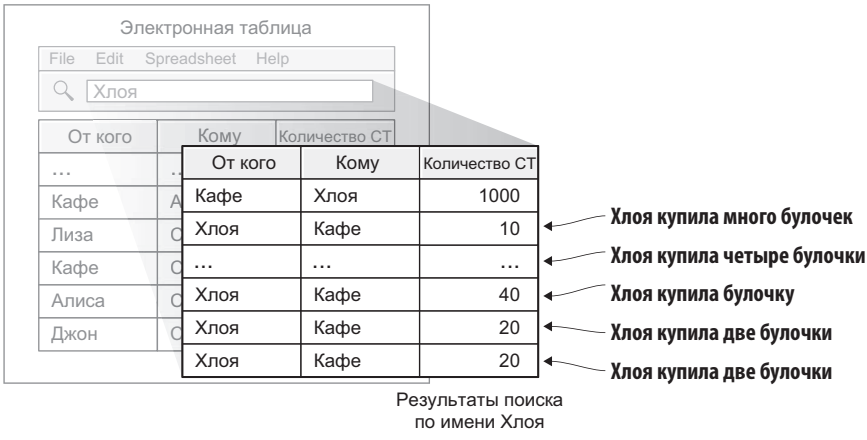
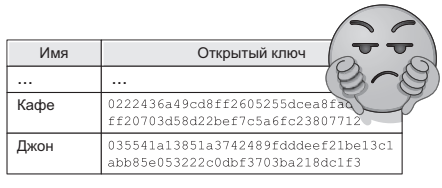


Рис. 3.2. Acme Insurances следит за привычкой Хлои есть много булочек

Замена имен открытыми ключами



Лиза постоянно обновляет таблицу имен и открытых ключей с тех пор, как сотрудники начали использовать цифровые подписи. Ей надоело это делать, поэтому она выступила с идеей, которая пойдет на пользу и ей, и всем остальным: заменить все имена в электронной таблице соответствующими открытыми ключами (рис. 3.3).

От кого	Кому	Количество СТ
…	…	…
Кафе	Анна	1000
Лиза	Кафе	10
Кафе	Компания	10 000
Алиса	Кафе	10
Джон	Кафе	10

От кого	Кому	СТ
…	…	…
0222436a49cd8ff2605255 dcea8fadd6a9aff20703d5 8d22bef7c5a6fc23807712	02b33f40f80812ae832404 e97e039eaa92e6993a6d14 7c8854c09281be1292e920	1000
036c4f8ed456142a75724d 57ab7f6c358850ee9a79dc 444fe5e754496c7cfa3371	0222436a49cd8ff2605255 dcea8fadd6a9aff20703d5 8d22bef7c5a6fc23807712	10
0222436a49cd8ff2605255 dcea8fadd6a9aff20703d5 8d22bef7c5a6fc23807712	037e944a7b778d190c05b5 9325c58eed069205148fa0 a2998273af0ffe36de9496	10 000
0317828d04ebd6d120e423 6bc0cf0cce12ebfdfa106c 7bf744deb547fcc52e768d	0222436a49cd8ff2605255 dcea8fadd6a9aff20703d5 8d22bef7c5a6fc23807712	10
035541a13851a3742489fd ddeef21be13c1abb85e053 222c0dbf3703ba218dc1f3	0222436a49cd8ff2605255 dcea8fadd6a9aff20703d5 8d22bef7c5a6fc23807712	10

Рис. 3.3. Замена имен открытыми ключами. Теперь разобраться в записях в электронной таблице намного сложнее, что хорошо скажется на конфиденциальности

Теперь трудно понять, сколько булочек съела Хлоя, не зная ее открытого ключа. Если сотрудники Acme Insurances получат копию этой новой электронной таблицы, они не смогут определить, кто является отправителями и получателями. Они будут видеть только открытые ключи отправителя и получателя каждого платежа.

Теперь Лиза может удалить свою громоздкую таблицу с именами и открытыми ключами. Но после этого пользователи больше не должны использовать имена при совершении платежей. Вместо этого они должны будут использовать открытый ключ отправителя и открытый ключ получателя (рис. 3.4).

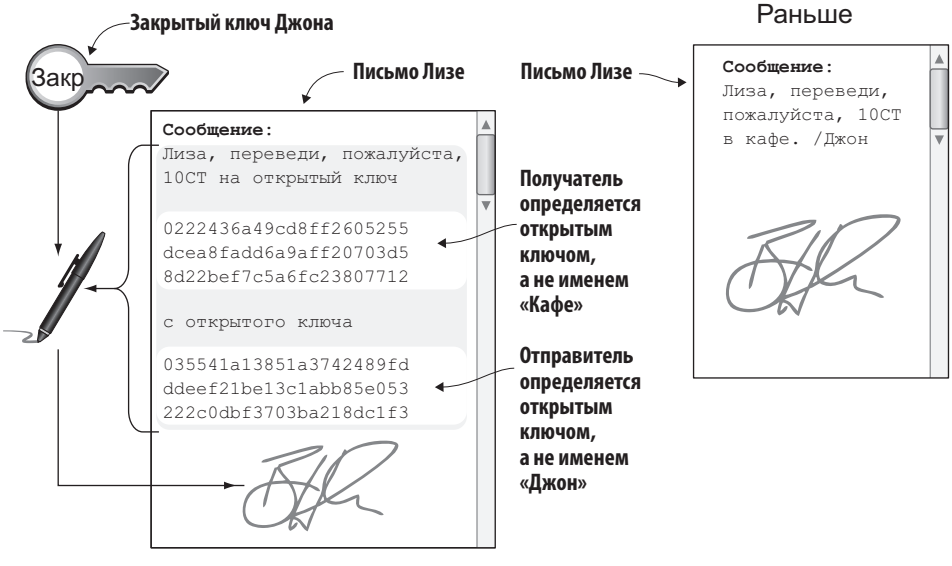


Рис. 3.4. В обновленной системе платежей вместо имен используются открытые ключи

Письмо Лизе содержит несколько важных элементов:

- Сообщение, включающее:
  - сумму;
  - открытый ключ отправителя;
  - открытый ключ получателя.
- Подпись, созданную закрытым ключом отправителя.

Главное отличие в том, что теперь платежи выполняются с использованием псевдонимов: имена заменили соответствующие открытые ключи. В остальном платеж выглядит так же, как и раньше.

Новая процедура проведения платежа

Предположим, на работу в компанию пришла новая сотрудница. Ее зовут Фаиза. Компания хочет послать ей 100 CT в качестве приветственного подарка. Как это сделать?

Для этого, во-первых, необходим открытый ключ получателя — Фаизы. Фаиза раньше не пользовалась системой жетонов, поэтому ей нужно создать пару ключей и передать открытый ключ отправителю — компании, — как показано на рис. 3.5.

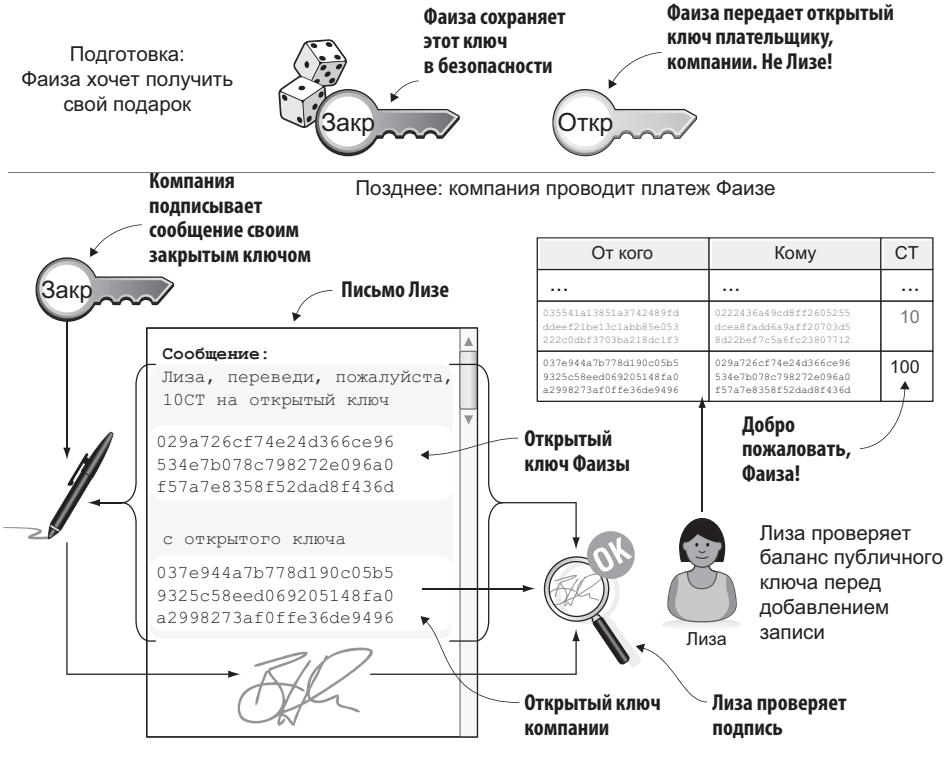


Рис. 3.5. Фаиза создает открытый ключ и передает его компании. Компания выполняет платеж на открытый ключ Фаизы

Фаиза создает закрытый и открытый ключи, следуя процедуре, что была описана в главе 2 в разделе «Улучшение безопасности жетонов на булочки», но пока не передает свой открытый ключ Лизе. Теперь, когда у Лизы нет таблицы имен и открытых ключей, нет смысла передавать ей открытый ключ. Ей это не нужно. Вместо этого Фаиза передает открытый ключ компании, которая хочет перевести ей жетоны на булочки.

Компания создает сообщение, в котором просит Лизу перевести 100 CT с `037e944a...36de9496` на `029a726c...ad8f436d`. Затем подписывает сообщение цифровой подписью и отправляет его Лизе. Лиза использует

- сообщение,
- открытый ключ отправителя
- и подпись,

чтобы убедиться, что сообщение подписано закрытым ключом отправителя, идентифицируемого открытым ключом. Она также проверяет баланс отправителя в электронной таблице. Делается это точно так же, как и во времена, когда в электронной таблице хранились имена, — она отыскивает записи, соответствующие открытому ключу отправителя, и вычисляет баланс.

ЛИЗА В БИТКОИН

Лиза решает те же задачи в отношении жетонов, что и майнер в Биткоин в отношении платежей в биткоинах.

От кого	Кому	CT
...	...	...
037e944a7b778d190c05b59325c58eed069205148fa0a2998273af0ffe36de9496	029a726cf74e24d366ce96534e7b078c798272e096a0f57a7e8358f52dad8f436d	100

Лиза прежде не видела открытый ключ получателя, но ей все равно. Ей важно лишь, чтобы отправитель имел деньги и сообщение было правильно подписано. Она переведет на баланс получателя сумму, указанную в сообщении.

Фаиза видит новую запись со своим открытым ключом в столбце «Кому», и это греет ей душу. Теперь она может тратить свои жетоны, как ей заблагорассудится. Фаизе не понадобилось беспокоить Лизу, посылая свой открытый ключ, и отвлекать ее от работы.

Итак, подведем итоги получившегося:

- имена в электронной таблице заменили открытые ключи;
- Лиза избавилась от таблицы с именами и открытыми ключами;

- платежи производятся с использованием открытых ключей отправителя и получателя вместо имен.

Эти изменения улучшили конфиденциальность и упростили работу Лизы. В конце этой главы мы подробно обсудим, как еще больше улучшить конфиденциальность.

В этом примере, получив письмо, *Лиза* почти наверняка будет знать, кто отправитель (в данном случае компания), посмотрев на поле «От» электронного письма. Но пока будем считать, что Лиза никому не раскрывает и никак не использует эту личную информацию. В этом примере мы используем электронную почту вместо одноранговой сети Биткоин. Сеть Биткоин, которая подробно обсуждается в главе 8, не использует личную информацию.

Подумайте, что теперь в Acme Insuranceans смогут узнать из электронной таблицы. Какую информацию они могут получить, если выяснят имя отправителя или получателя *одного* платежа? Они смогут идентифицировать все платежи, которые сделал человек.

Укорачивание открытых ключей

Использование открытых ключей в электронной таблице улучшило конфиденциальность, но такие ключи занимают много места по сравнению с именами. Имя «Джон» занимает 4 байта в электронной таблице, тогда как открытый ключ — 33 байта. Важно, чтобы электронная таблица была как можно меньше, потому что чем меньше электронная таблица, тем быстрее сотрудники смогут ее загружать, желая проверить свой баланс, и тем меньше места она будет занимать на жестком диске Лизы.

Хеширование открытого ключа до 20 байт

Среди сотрудников нашлись такие, кто решил, что смогут заменить открытые 33-байтные ключи чем-то более коротким без особого ущерба для безопасности. Они предложили заменить каждый открытый ключ в электронной таблице жетонов криптографическим хешем открытого ключа. Это сокращает объем данных в столбцах «От кого» и «Кому» в электронной таблице, а также защищает деньги пользователей в случае ошибки в функции вычисления открытого ключа, как будет показано ниже. Хеширование выполняется не одной, а двумя разными криптографическими хеш-функциями, как показано на рис. 3.6. Мы обсудим причину использования двух хеш-функций в следующем разделе.

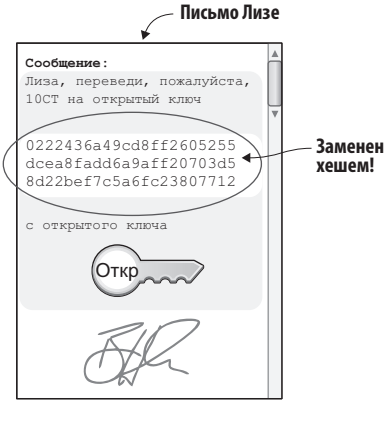
От кого	Кому	СТ
…	…	…
035541a13851a3742489fd ddeef21be13c1abb85e053 222c0dbf3703ba218dc1f3	0222436a49cd8ff2605255 dcea8fadd6a9aff20703d5 8d22bef7c5a6fc23807712	10
037e944a7b778d190c05b5 9325c58eed069205148fa0 a2998273af0ffe36de9496	029a726cf74e24d366ce96 534e7b078c798272e096a0 f57a7e8358f52dad8f436d	100

От кого	Кому	СТ
…	…	…
5f2613791b36f667fdb8 e95608b55e3df4c5f9eb	87e3d1692022a7744bf2 406a963c656c8393b1cc	10
bc27a2f538aa6a796e4b 2197f150ae0f667870eb	bea73261a7499c22f8e1 e57bdb0e41ffc35ce56a	100

Рис. 3.6. Замена открытого ключа хешем RIPEMD160 от хеша SHA256 открытого ключа

Открытый ключ сначала хешируется с помощью функции SHA256, с которой вы познакомились в предыдущей главе. Затем результат этой криптографической хеш-функции хешируется с помощью RIPEMD160 — криптографической хеш-функции, которая возвращает 160-битное (20-байтное) число. Этот окончательный хеш мы будем называть *хешем открытого ключа* (Public Key Hash, PKH). Все открытые ключи в электронной таблице были заменены соответствующими хешами PKH.

КАК ВЫПОЛНЯЛИСЬ ПЛАТЕЖИ РАНЬШЕ



Теперь процесс оплаты отличается от того, который имел место, когда Фаиза получила свои 100 CT от компании. Предположим, Джон решил купить булочки (рис. 3.7).

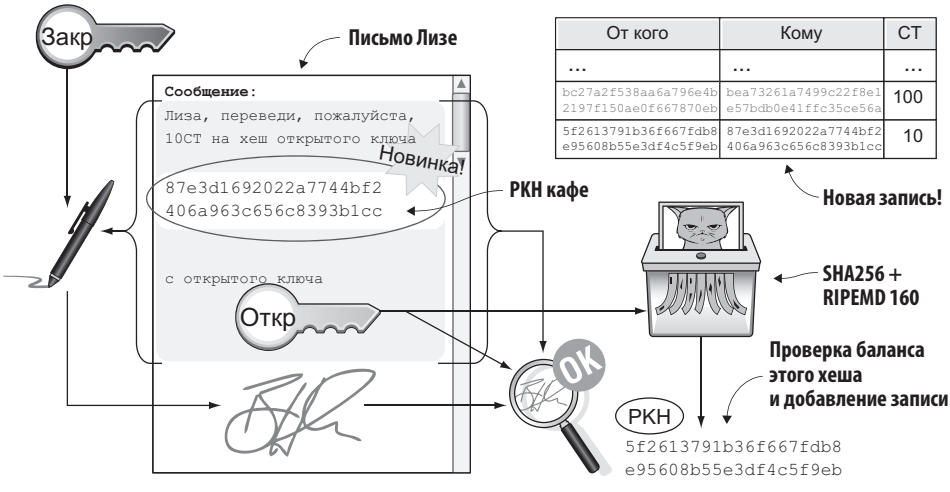


Рис. 3.7. Джон покупает булочку. Отправитель все еще представлен открытым ключом, но получатель — хешем открытого ключа РКН. Лизе нужно создать РКН из открытого ключа отправителя, чтобы проверить его баланс и выполнить платеж

Во-первых, немного изменилось сообщение Лизе. В качестве получателя Джон должен указать хеш РКН кафе вместо открытого ключа. Отправитель по-прежнему представлен открытым ключом, поскольку он необходим для проверки подписи. Лиза больше не хранит открытые ключи людей.

Во-вторых, поскольку теперь электронная таблица содержит хеши РКН, Лиза должна вычислять РКН для открытого ключа отправителя, чтобы проверить его баланс и добавить запись о платеже в электронную таблицу.

Почему используются SHA256 и RIPEMD160?

Использование RIPEMD160 в роли последней криптографической хеш-функции является осознанным выбором для укорачивания РКН. Сравните вывод SHA256 с выводом RIPEMD160:

- SHA256:

```
85ae273f0aa730eddf2285d3f3ab071eb29caba1e428db90e6dfbd71b8e1e918
```

- RIPEMD160:
  `5f2613791b36f667fdb8e95608b55e3df4c5f9eb`

Такое решение обеспечивает хороший баланс между безопасностью и размером.

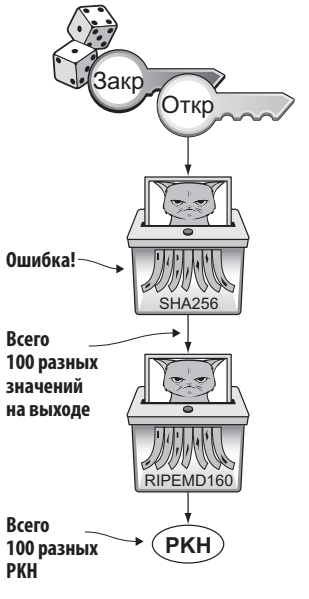
Но почему используются две разные криптографические хеш-функции? Мы не знаем причин выбора этой схемы в системе Биткоин, потому что ее изобретатель, Сатоши Накамото, прекратил переписку с сообществом Биткоин. Но давайте не будем строить догадки, а обсудим некоторые свойства этой схемы.

P2PKH

Большинство платежей в системе Биткоин осуществляется с использованием PKH в качестве получателя. Платежи этого типа часто называют *платежами на хеш открытого ключа* (pay-to-public-key-hash, p2pkh), но существуют и другие типы платежей.

Если одна из хеш-функций окажется нестойкой к атакам по определению исходного образа, другая добавит сложности. То есть если кто-то научится вычислять исходные данные по хешу RIPEMD160, ему придется выполнить атаку по определению исходного образа для хеша SHA256 (выполнить еще примерно 2^{255} попыток), чтобы найти открытый ключ. Аналогично, если кто-то научится вычислять исходные данные по хешу SHA256, ему сначала придется выполнить атаку по определению исходного образа для хеша RIPEMD160 и только потом использовать полученный результат для вычисления открытого ключа.

Вместе с тем, если длина хеша, возвращаемого любой криптографической хеш-функцией, меньше, это ухудшает безопасность объединенной цепочки хеш-функций. Чтобы было понятнее, представьте, что SHA256 имеет только 100 возможных выходных значений. Вы можете украсть деньги у кого угодно, попробовав разные случайные закрытые ключи и вычислив соответствующие PKH. Найдя PKH, соответствующий искомому, вы найдете закрытый ключ, с помощью которого можно украсть деньги. В среднем достаточно проверить всего 50 разных закрытых ключей, чтобы украсть деньги с одного PKH. Это свойство вбирает худшее из обоих миров: если одна из двух функций окажется ненадежной, то и вся цепочка будет ненадежной. Вероятность, что любая функция имеет этот недостаток, мала. Если такой недостаток су-



ществует, то выгоды от укорачивания выходного набора, вероятно, окажутся недостаточно значительными, чтобы поставить под угрозу безопасность. Напомню, что пока не удалось найти ни одной коллизии в этих криптографических хеш-функциях.

Следует также отметить, что эти две криптографические хеш-функции были разработаны разными организациями. RIPEMD160 была разработана в Европейском университете в сотрудничестве с широким сообществом криптографов. SHA256 была разработана Агентством национальной безопасности США (National Security Agency, NSA). Обе считаются безопасными, и обе подвергались проверке со стороны большого числа людей.

Теперь, повысив безопасность таблицы жетонов, поразмышляем о конфиденциальности. Улучшило ли это конфиденциальность? Сложнее ли Acme Insurances выяснить информацию о том, кто кому платит, по сравнению с тем, когда в таблице использовались открытые ключи?

КОНФИДЕНЦИАЛЬНОСТЬ УЛУЧШИЛАСЬ?

Нет.

Увы, нет. Открытые ключи и РКН связаны почти однозначным соответствием. Использование РКН скрывает личную информацию не больше, чем использование простых открытых ключей.

Избегание дорогостоящих опечаток

Когда Лиза проверяет платеж перед выполнением, ей все равно, кто является получателем или даже является ли он существующим получателем. Она вставит в столбец «Кому» таблицы то, что укажет плательщик. Она даже не знает, является ли получатель действительным, потому что не знает всех открытых ключей.

Это удобно для Лизы, но люди могут терять деньги из-за неосторожности. Представьте еще раз, что Джон хочет купить булочку. Но на этот раз он был недостаточно внимателен и допустил опечатку в сообщении, как показано на рис. 3.8.

Он допустил опечатку в хеше РКН получателя. В последнем символе хеша он ввел `d` вместо `c`. Что теперь произойдет?

Джон не заметил ошибки и в предвкушении вкусной булочки подписывает сообщение и отправляет электронное письмо Лизе. Лиза проверяет подпись и вычисляет РКН отправителя. Ее не интересует получатель. Она добавляет

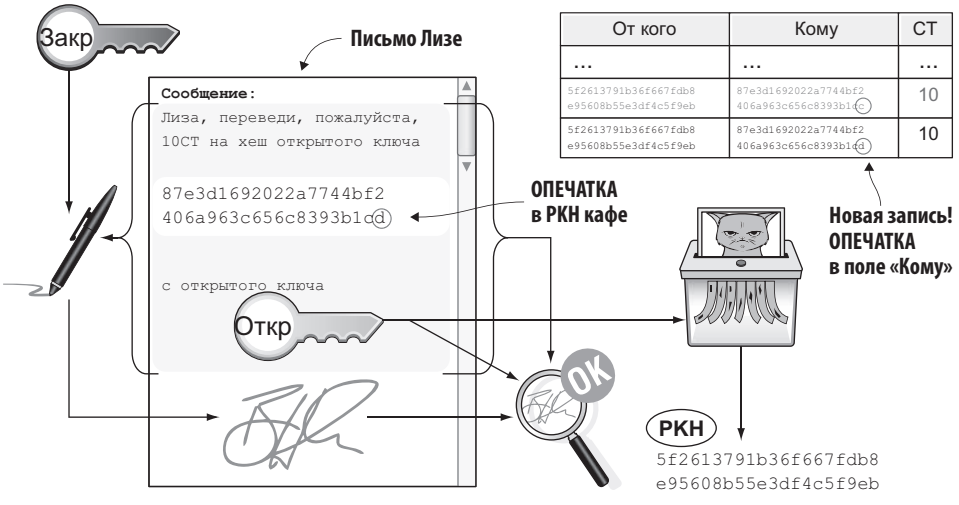
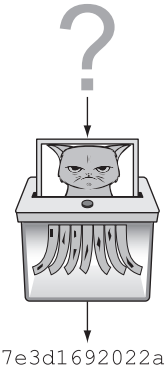


Рис. 3.8. Джон допустил опечатку в хеше РКН получателя. И что теперь?

новую запись в таблицу, выполняя перевод от плательщика `5f261379...f4c5f9eb` получателю `87e3d169...8393b1cd`.

> **ПРОВЕРКА ПОЛУЧАТЕЛЯ НЕ ВЫПОЛНЯЕТСЯ**
>
> Нет такого понятия, как «неверный» РКН получателя. Лиза выполнит перевод любому получателю, если электронная подпись окажется верна.

На этом ее обязанности заканчиваются, и она переходит к другим делам. Владелец кафе выполняет поиск своего хеша РКН в таблице, но не находит входящий платеж. Джон стоит у стойки в кафе и с возмущением требует «отдать ему эту чертову булочку», потому что он *перевел* деньги. Владелец кафе отказывается. Джон внимательно смотрит на электронную таблицу и ищет свой РКН. Он находит перевод, который только что сделал, и понимает, что допустил опечатку.



Джон отправил деньги на «хеш открытого ключа» (РКН), для которого нет известного закрытого ключа. Никто и никогда не сможет потратить эти 10 CT — ни кафе, ни Джон, ни кто-либо еще. Джон только что сжег 10 CT.

К сожалению, в будущем такое может случаться снова и снова, если ничего не сделать, чтобы предотвратить это. Проблема может возникнуть на любом этапе: когда

владелец кафе записывает свой РКН, чтобы передать его Джону, когда Джон пишет сообщение Лизе. Нетрудно представить, что Лиза тоже может допустить подобную ошибку, добавляя запись в электронную таблицу, хотя это и не наш случай, потому что Лиза настолько скрупулезно выполняет свои обязанности, что подобное просто *невозможно*. Она лучше всех делает свою работу. Лиза никогда не станет причиной сгорания чужих средств.

На чем мы остановились?

В этой главе мы обсуждаем адреса в системе Биткоин. Чтобы напомнить вам, где используются адреса в Биткоин, еще раз приведем на рис. 3.9 фрагмент диаграммы из главы 1.

Рис. 3.9. Адреса в Биткоин

К концу этой главы мы получим биткоин-адреса (адреса жетонов на булочки). Мы только что заменили имена в электронной таблице хешами открытых ключей — РКН. Теперь перейдем к *биткоин-адресам*. Биткоин-адрес — это *преобразованный РКН*, то есть РКН, записанный в более удобочитаемой для человека форме, защищающей от опечаток. Теперь Лизе (узлу Биткоин) будет посылаться не РКН, а адрес — строка, которую пользователи видят и сообщают друг другу.

Base58check

Сотрудники, обеспокоенные безопасностью, обсудили проблему с опечатками и предложили идею *адресов жетонов на булочки*. Адрес жетона — это РКН, *закодированный* так, что позволяет обнаружить ошибку ввода. РКН легко можно преобразовать в закодированную форму и обратно.

Предположим, Фаиза пожалела Джона и решила отдать ему 20 CT из своих 100 CT, чтобы облегчить его боль. Она боится совершить ту же ошибку, что и Джон, поэтому спрашивает его адрес жетонов. Джон создает его, кодируя свой PKH с помощью функции *base58check* (рис. 3.10).

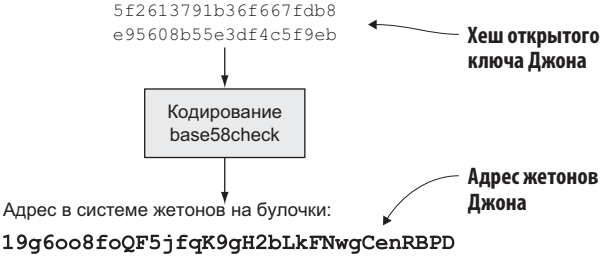


Рис. 3.10. Схема кодирования base58check для преобразования PKH в адрес жетонов на булочки

АДРЕСА В БИТКОИН

Адреса в системе жетонов на булочки совпадают с наиболее распространенными версиями адресов в Биткоин. Однако есть и другие типы биткоин-адресов.

В результате получается адрес Джона в системе жетонов на булочки: `19g6oo8f...gCenRBPD`. Джон передает этот адрес Фаизе, которая затем переводит жетоны Джону, действуя, как показано на рис. 3.11.

Процедура оплаты меняется для плательщика, но ничего не меняется для Лизы. Фаиза проверит адрес Джона, *декодировав* его в PKH. Успешное декодирование гарантирует, что адрес не содержит опечаток.

Как упоминалось выше, PKH можно преобразовать в адрес и обратно. Это не однонаправленная функция. Это просто разные способы представления PKH в виде последовательности байтов или адреса (рис. 3.12).

КТО БУДЕТ ПОЛЬЗОВАТЬСЯ АДРЕСАМИ В СИСТЕМЕ ЖЕТОНОВ НА БУЛОЧКИ?

Адреса жетонов предназначены только для безопасной передачи PKH между пользователями. Лиза не видит их.

Содержимое письма Лизе не изменилось. Адреса используются только пользователями. Лиза никак не использует их для проверки и не заносит в электронную таблицу.

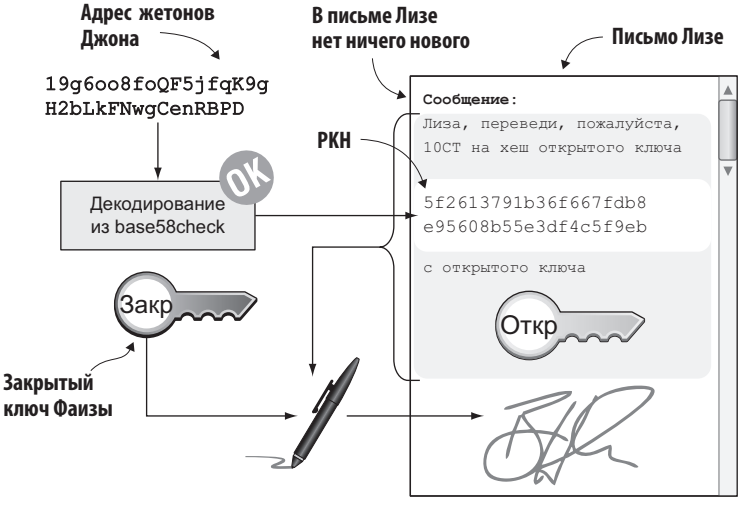


Рис. 3.11. Фаиза выполняет перевод на адрес Джона. Она декодирует адрес в РКН и проверяет отсутствие опечаток

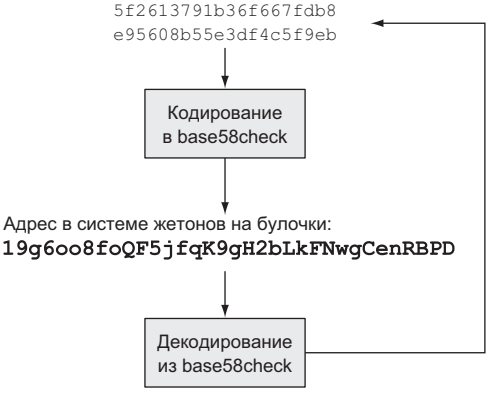


Рис. 3.12. РКН можно закодировать в адрес и обратно декодировать из адреса

Кодирование base58check

Давайте посмотрим, как работает эта загадочная кодировка base58check (рис. 3.13). Во-первых, она добавляет номер версии перед РКН. Люди, придумавшие идею адресов, хотели упростить будущие обновления формата адреса. На данный момент доступна только одна версия адресов, которой соответствует 1-байтный номер 0.

Для обнаружения опечаток добавляется *контрольная сумма*. Она рассчитывается из РКН с номером версии. Чтобы получить контрольную сумму,

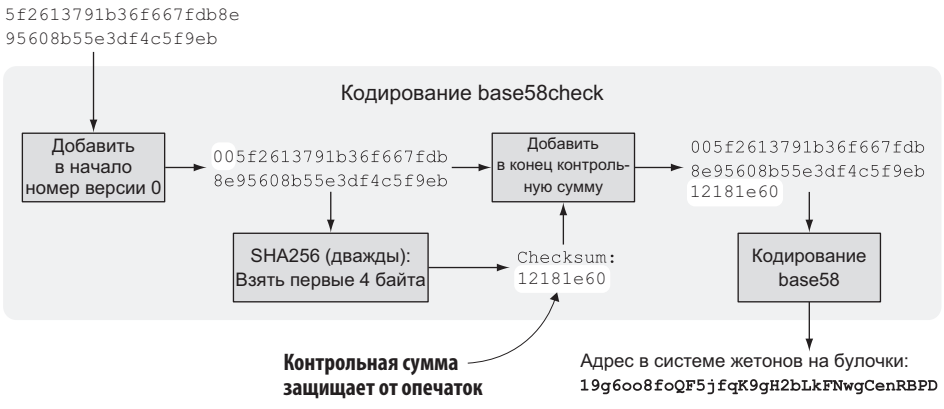
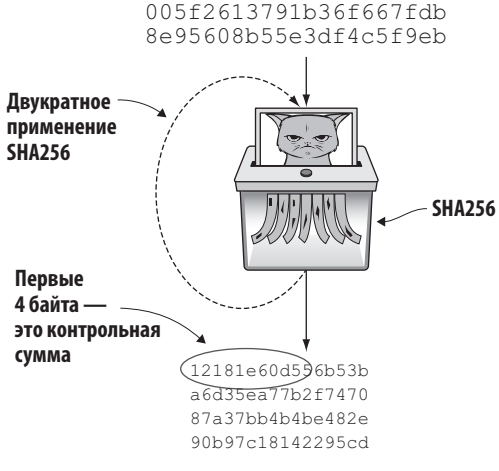


Рис. 3.13. Кодирование base58check хеша открытого ключа Джона. Сначала создается строка, включающая номер версии, хеш и контрольную сумму. Затем строка кодируется в формат base58

base58check хеширует PKH с номером версии, дважды применяя SHA256. Это означает, что сначала вычисляется хеш PKH, после чего полученный хеш хешируется повторно. Далее из второго хеша извлекаются первые 4 байта и добавляются к PKH с номером версии как контрольная сумма. Как эта контрольная сумма защищает пользователей от опечаток, вы увидите чуть ниже. А пока наберитесь терпения!



Мы начали с хеша открытого ключа (PKH) размером 20 байт (40 шестнадцатеричных символов). Затем мы добавили номер версии и контрольную сумму и получили 25 байт (50 шестнадцатеричных символов). Чтобы компенсировать это увеличение, закодируем 25 байт более компактным способом, чем позволяет шестнадцатеричное кодирование.

Более компактный способ кодирования

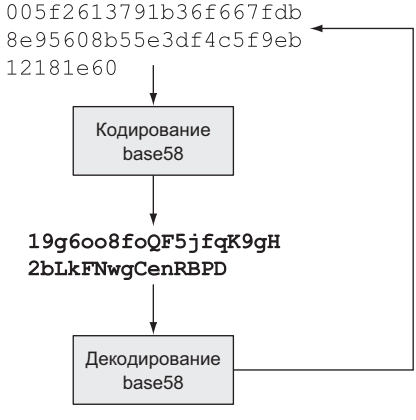
Шестнадцатеричное кодирование — не самый эффективный способ представления байтов данных. Для представления каждого байта требуется

2 символа. В этом случае используется только 16 символов, каждый из которых представляет 4 бита, от `0000` до `1111`.

Существует много схем кодирования, использующих больше символов для представления данных. Наиболее широко известна схема base64, в которой каждый символ представляет 6 бит данных; для этого в схеме используется следующий алфавит, включающий не только буквы и цифры:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

Символ `A` представляет биты `000000`, `B` представляет `000001`, а символ `/` представляет `111111`. Это простой и компактный способ представления данных с помощью читаемых человеком символов. В этой книге я уже несколько раз использовал данные в кодировке base64 для представления подписей.



Но base64 не очень удобна для кодирования адресов жетонов на булочки. Нам нужна кодировка, которая не просто обнаруживает ошибки ввода, когда они происходят, но и уменьшает риск их появления. Обратите внимание, что в некоторых шрифтах некоторые символы выглядят одинаково, например, `lI` (строчная буква `L`, заглавная `I`) и `0O` (ноль и заглавная `O`). Нам также нужен формат, упрощающий копирование и вставку, не содержащий специальных символов, таких как `+` и `/`, которые не позволяют выделить адрес целиком двойным щелчком мыши. Отказ от этих шести символов уменьшит вероятность опечаток. Но теперь осталось только 58 символов, поэтому нам нужен другой тип кодирования.

Этот новый тип кодирования называется *base58*, потому что его алфавит состоит из следующих 58 символов:

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

ВНИМАНИЕ

Если почувствуете, что начинаете путаться в этих низкоуровневых деталях base58, переходите сразу к разделу «Декодирование base58check» и просто считайте, что base58 — это способ кодирования и декодирования данных. Остальные могут продолжать чтение, это довольно интересно.

В base64 каждый символ представляет ровно 6 бит, что упрощает кодирование и декодирование данных. Но в base58 каждый символ представляет чуть менее 6 бит, но более 5 бит. Поэтому данные должны кодироваться иначе.

Вернемся к примеру, в котором Джон создает свой адрес. Он только что добавил версию и контрольную сумму. Теперь пришло время закодировать 25 байт в адрес (рис. 3.14).

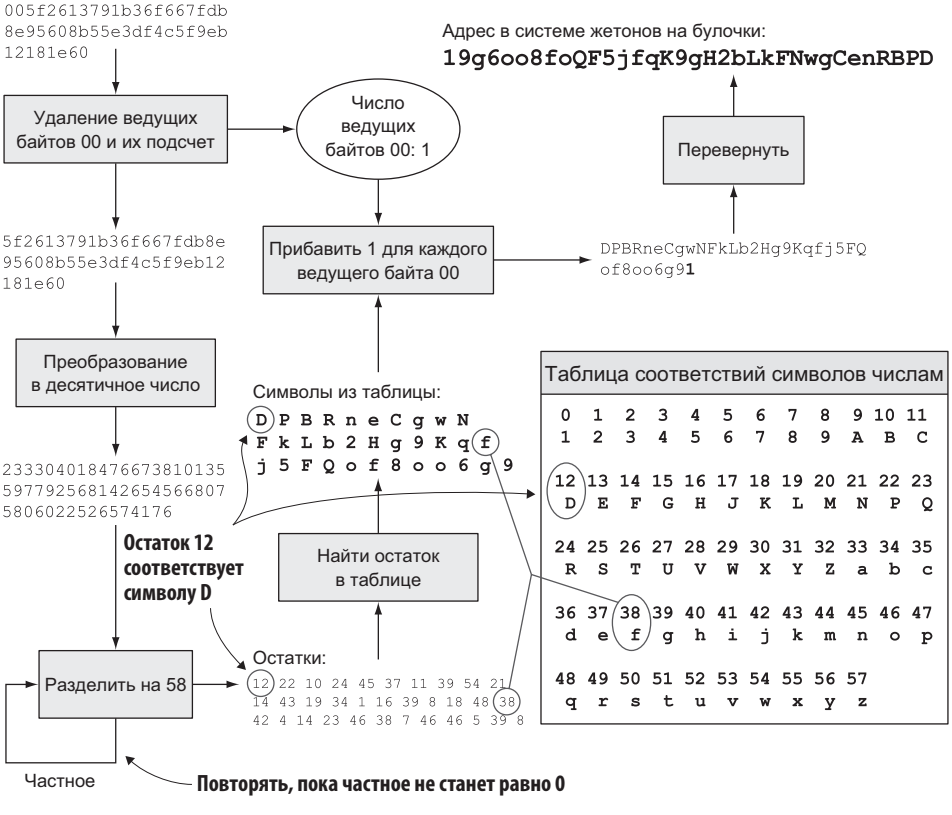


Рис. 3.14. Кодирование хеша открытого ключа Джона с номером версии и контрольной суммой методом base58. Главным здесь является деление числа на 58 и запоминание остатков, которые затем отображаются в символы с помощью таблицы

Стратегия base58 основана на интерпретации данных как огромного числа, которое снова и снова делится на 58, пока частное не станет равным 0, с запоминанием остатков от каждого деления. Затем выполняется поиск каждого остатка в таблице и в конец добавляется `1` для каждого ведущего байта `00` во

входных данных. Затем полученная строка переворачивается, и в результате получается адрес Джона в системе жетонов на булочки. Обратите внимание, что все адреса, не только Джона, будут начинаться с `1`. Это объясняется тем, что байт с номером версии 0 кодируется символом `1`.

Данные в кодировке base58, такие как адрес Джона, можно декодировать обратно в исходное значение. Я оставлю это как самостоятельное упражнение тем, кому это будет интересно.

Обратите внимание, что в кодировке base58 нет ничего нового. Она использует обобщенный алгоритм преобразования десятичного числа в систему счисления с любым другим основанием. Этот алгоритм с тем же успехом можно использовать для преобразования в base3, выполняя деление на 3 вместо 58. При этом, возможно, вы захотите изменить таблицу и отобразить 0 в 0, 1 в 1 и 2 в 2, чтобы получить привычные символы. Например, запишем число 17 в кодировке base3:

$17 = 5 \times 3 + 2$

$5 = 1 \times 3 + 2$

$1 = 0 \times 3 + 1$

Затем найдите остатки в таблице поиска (те же цифры, что и преобразованные), и вы получите `2 2 1`. Переверните эту последовательность, чтобы получить конечный результат: `1 2 2`. Убедитесь в его правильности, как показано ниже:

$1 \times 3^2 + 2 \times 3^1 + 2 \times 3^0 = 9 + 6 + 2 = 17$

Декодирование из формата base58check

Джон только что создал свой адрес в системе жетонов, применив кодирование base58check к своему РКН. Он дал адрес Фаизе, чтобы она могла отправить ему 20 СТ. Теперь Фаиза должна написать сообщение Лизе. Для этого ей нужен РКН Джона. Кодирование base58check обладает одним замечательным свойством — это обратимый процесс, то есть из адреса можно получить РКН и одновременно проверить наличие опечаток (рис. 3.15).

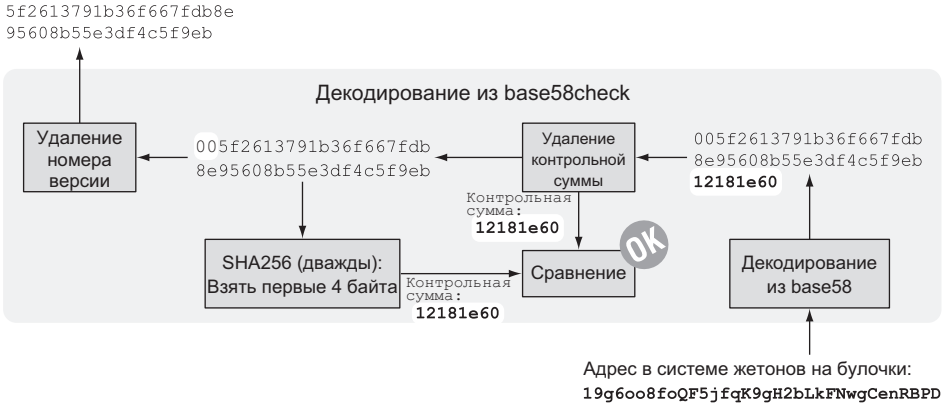


Рис. 3.15. Декодирование base58check производится выполнением обратных операций кодирования base58check. Опечатки обнаруживаются сравнением контрольных сумм

Фаиза берет адрес Джона и выполняет его base58-декодирование. Затем удаляет контрольную сумму и использует оставшуюся часть, хеш открытого ключа РКН с номером версии, чтобы снова вычислить контрольную сумму. Вновь вычисленная и только что удаленная контрольные суммы должны совпадать. Если это не так, значит, в адресе присутствует ошибка и тогда Фаиза не создаст сообщение. Она поймет, что на каком-то этапе адрес был поврежден, и воздержится от отправки письма Лизе. Она проверит, правильно ли она сама ввела адрес и правильный ли адрес передал ей Джон, чтобы узнать, где возникла ошибка.

Насколько надежна такая защита с контрольной суммой? Предположим, что при вводе адреса допущена опечатка. Какова вероятность того, что контрольная сумма не обнаружит ошибку? Контрольная сумма составляет 4 байта, что соответствует $2^{32} \approx 4{,}3$ миллиарда значений. Вероятность, что base58check не обнаружит ошибку, составляет примерно 1 на 4,3 миллиарда. То есть защита довольно надежна.

Возвращаемся к конфиденциальности

Замена имен на РКН определенно повысила уровень конфиденциальности, но в электронной таблице все еще есть информация, которую Acme Insurances может счесть полезной.

Например, в Acme наверняка смогут определить, что РКН `87e3d169...8393b1cc` принадлежит кафе, потому что на него делается много платежей на сумму

10 CT. Далее, Acme сможет выяснить, какие РКН производят больше всего платежей по 10 CT на этот РКН. Допустим, сотрудник Acme в беседе с Фаизой попросил рассказать о ее недавних платежах. Пока она сделала только один платеж — перевела жетоны Джону. Фаиза, не зная, почему Acme задает эти вопросы, рассказала, что жетоны переведены Джону.

ЭКСПЕРТИЗА

Этот прием часто используется в Биткоине, например, в расследовании уголовных преступлений.

Неделю спустя Джон получает письмо от Acme, где ему вежливо сообщают, что он попал в группу риска и его страховая премия была соответствующим образом скорректирована.

Уважаемый Джон,

> Нам стало известно, что вы ведете нездоровый образ жизни. Поэтому мы перевели вас в категорию повышенного риска.
>
> Поздравляем.
>
> С уважением, Acme Insurances

Очевидно, что некоторые проблемы конфиденциальности все же остаются. К счастью, пользователи могут создавать сколько угодно адресов. Например, кафе может создать уникальный адрес для каждого входящего платежа. И Джон сможет создать новый адрес, когда в следующий раз Фаиза решит поделиться с ним своими жетонами.

Использование уникальных адресов для каждого платежа усложнит для сотрудников Acme извлечение информации из электронной таблицы, поскольку они не смогут определить, какие платежи принадлежат конкретному лицу.

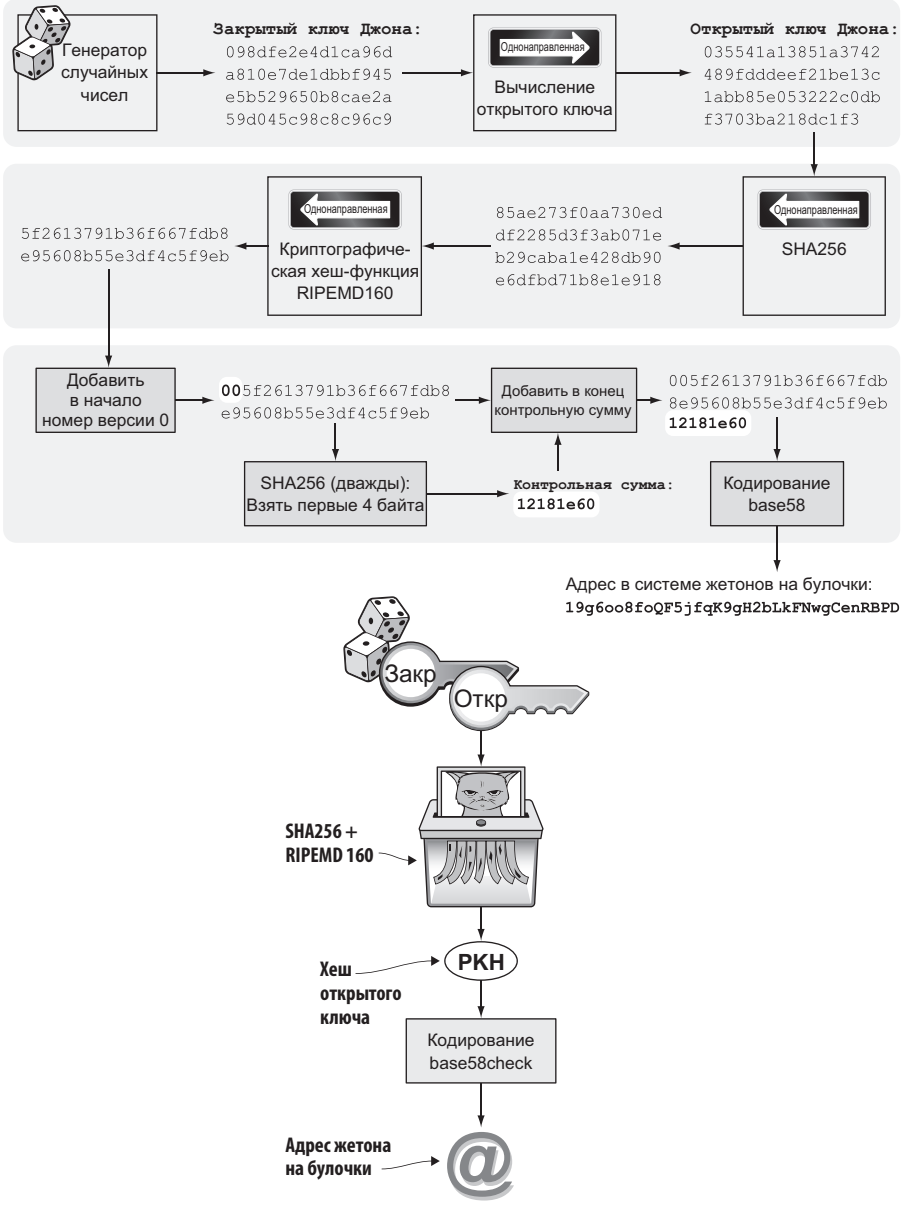
Повторение

Эта глава началась с замены имен пользователей соответствующими хешами открытых ключей.

От кого	Кому	CT
…	…	…
Джон	Кафе	10
Компания	Фаиза	100

От кого	Кому	CT
…	…	…
5f2613791b36f667fdb8 e95608b55e3df4c5f9eb	87e3d1692022a7744bf2 406a963c656c8393b1cc	10
bc27a2f538aa6a796e4b 2197f150ae0f667870eb	bea73261a7499c22f8e1 e57bdb0e41ffc35ce56a	100

Затем мы использовали метод кодирования base58check для создания адреса из PKH. Давайте соберем вместе все, что мы узнали, и изобразим весь процесс от генератора случайных чисел до адреса.



Прежде чем подписать свое сообщение, Фаиза проверяет отсутствие опечаток, выполняя декодирование base58check адреса.

Изменения в системе

Таблица понятий (табл. 3.1) не изменилась в этой главе. Адреса, используемые в системе жетонов на булочки, в точности совпадают с адресами в системе Биткоин, поэтому мы не вводили никаких новых понятий, отличающихся от понятий в Биткоин.

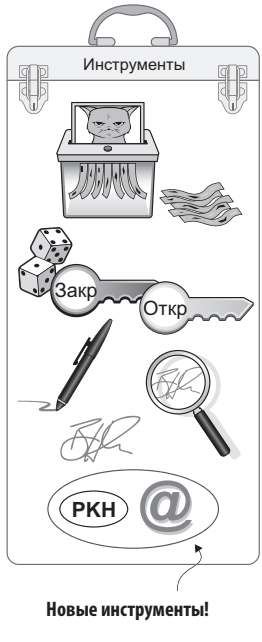


Таблица 3.1. Ничего нового в таблице понятий

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
Электронная таблица	Блокчейн	Глава 6
Письмо Лизе	Транзакция	Глава 5
Запись в электронной таблице	Транзакция	Глава 5
Лиза	Майнер	Глава 7

Благодаря РКН и адресам в системе жетонов на булочки, которые вы можете добавить в свой набор инструментов, Лиза может отказаться от таблицы открытых ключей. И теперь мы выпускаем версию 3.0 системы жетонов на булочки (табл. 3.2).

Таблица 3.2. Примечания к релизу, система жетонов на булочки 3.0

Версия	Особенность	Как реализована
НОВАЯ 3.0	Защищенность от дорогостоящих опечаток	Адреса в системе жетонов на булочки
	Улучшенная конфиденциальность	Вместо имен в электронной таблице сохраняются хеши открытых ключей (РКН)
2.0	Защищенная платежная система	Цифровые подписи решают проблему самозванцев
1.0	Простая платежная система	Основывается на доверии Лизе и личных знакомствах
	Конечный объем денег	Лиза получает в награду 7200 СТ ежедневно; величина вознаграждения уменьшается вдвое каждые четыре года

Упражнения

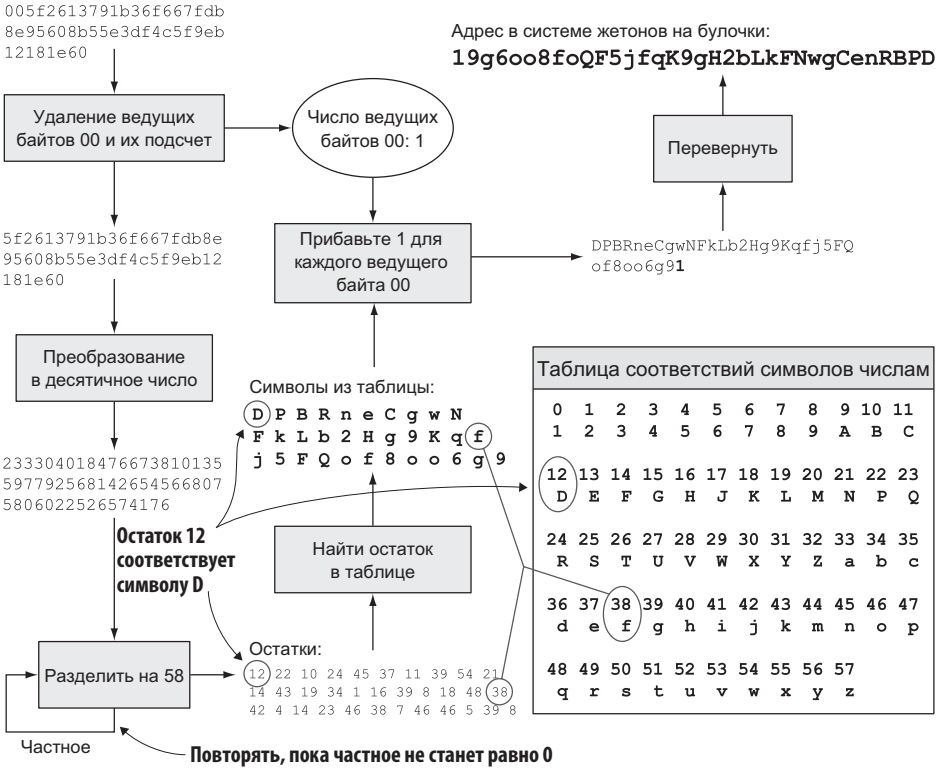
Для разминки

3.1. РКН короче открытого ключа — всего 160 бит. Мы сделали его короче, использовав RIPEMD160. Почему мы решили укоротить ключ? Тому есть две веские причины.

3.2. Для создания адреса в системе жетонов на булочки (Биткоин) из РКН используется кодирование base58check. Существует ли обратное преобразование, позволяющее получить РКН по адресу?

3.3. Когда и кем используется декодирование base58check?

3.4. Закодируйте два шестнадцатеричных байта 0047 методом base58. Используйте схему, представленную ниже. Вы можете пропустить это упражнение, если не читали раздел о кодировании base58.



3.5. Какая часть адреса защищает его от опечаток?

Придется пораскинуть мозгами

3.6. Представьте, что Джон решил купить булочку из кафе. У него два адреса: $@_1$ с 5 СТ и $@_2$ с 8 СТ. Общий баланс составляет 13 СТ, поэтому он может потратить 10 СТ за булочку. Приведите пример, как он мог бы перевести 10 СТ в кафе.

3.7. Можно ли определить, какие адреса связаны с определенным платежом, просмотрев следующую таблицу?

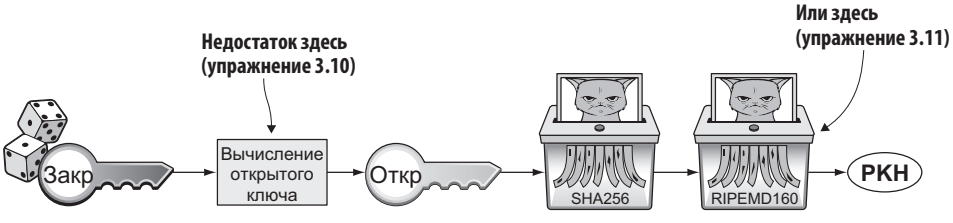
От кого	Кому	СТ
...	...	...
5f2613791b36f667fdb8 e95608b55e3df4c5f9eb	87e3d1692022a7744bf2 406a963c656c8393b1cc	10
...	...	...

Можно определить адрес?

3.8. Можно ли определить, какие открытые ключи задействованы в определенном платеже, просто заглянув в электронную таблицу?

3.9. Предположим, что для всех платежей всегда используются уникальные адреса. Какую информацию из электронной таблицы сможет использовать Acme для примерного определения адресов, принадлежащих кафе?

3.10. Предположим, что в функции создания открытого ключа имеется серьезный недостаток, из-за чего любой сможет вычислить закрытый ключ на основе открытого. Что мешает злоумышленнику украсть ваши деньги в этом сценарии?



3.11. Предположим, в функции RIPEMD160 имеется серьезный недостаток, из-за чего любой сможет легко вычислить 256-битный исходный образ PKH. Это означает, что функция неустойчива к восстановлению исходного образа. Что мешает злоумышленнику украсть ваши деньги в этом сценарии?

Итоги

- Конфиденциальность важна не только для преступников, но и для обычных людей.
- Использование хеша открытого ключа (PKH) в качестве получателей платежей вместо личных имен важно для обеспечения конфиденциальности и безопасности.
- Кодирование PKH в адреса Биткоин или адрес жетонов на булочки снижает риск отправки денег в пустоту благодаря контрольной сумме в адресе.
- Адреса Биткоин представляют интерес только для пользователей Биткоин. Сеть Биткоин или Лиза работают с простыми PKH.
- Вы можете иметь сколько угодно адресов Биткоин. Использование нескольких адресов, например по одному на каждый полученный платеж, повышает конфиденциальность сделок.

4 Кошельки

Эта глава охватывает следующие темы:

- ✓ автоматизация платежей;
- ✓ создание ключей и управление ими;
- ✓ простота создания надежно защищенных резервных копий ключей.

Пока что мы ничего не сделали, чтобы сотрудникам компании было проще и удобнее использовать систему жетонов на булочки. Между тем ситуация усложнилась, потому что теперь пользователям приходится добавлять в письма Лизе больше информации, чем вначале. Кроме того, пользователи вынуждены предпринимать дополнительные шаги, чтобы использовать несколько адресов ради своей конфиденциальности.

В этой главе мы сконструируем мобильное приложение — *кошелек* (рис. 4.1), которое поможет пользователям автоматизировать решение многих распространенных задач. Кошелек будет создавать новые адреса, хранить закрытые ключи, упрощать передачу адресов между пользователями и автоматизировать процесс перевода платежей.

Мы обсудим разные подходы к резервному копированию содержимого кошелька. Рассмотрим новый способ генерации ключей — *иерархически детерминированные кошельки* (hierarchical deterministic wallets), или HD-кошельки, чтобы еще больше упростить резервное копирование; он требует создать резервную копию единственного случайного числа, которое называется *начальным числом*. И в конце главы для интересующихся будет представлен подробный обзор математических основ создания открытого ключа.

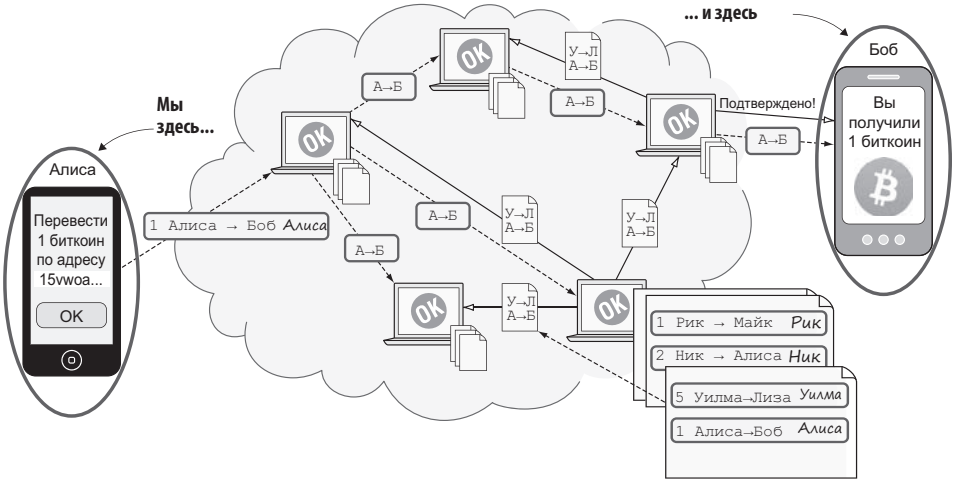


Рис. 4.1. Использование кошелька Биткоин

Эта глава ничего не изменит в работе Лизы или таблицы. В центре нашего внимания будут находиться только пользователи.

Первая версия кошелька

Группа разработчиков программного обеспечения в нашей компании взялась за создание мобильного приложения под названием *кошелек*, чтобы упростить для себя и других пользователей решение типичных задач, к числу которых они отнесли:

- *Создание новых адресов* — пользователи должны время от времени создавать новые адреса в системе жетонов на булочки. Разные адреса можно использовать для разных целей или даже для отдельных платежей, из соображений конфиденциальности и безопасности.
- *Управление закрытыми ключами* — для каждого созданного адреса кошелек должен хранить соответствующий закрытый ключ и управлять им. Сохранение закрытых ключей в безопасности от злоумышленников является сложной задачей.

КОШЕЛЬКИ БИТКОИН

В настоящее время доступно несколько кошельков для Биткоин. К наиболее популярным относятся:

- Bitcoin Core;
- Electrum;
- GreenBits;
- BRD (Bread).

Исчерпывающий список вы найдете на веб-ресурсе 10 (см. приложение В).

- *Передача плательщику платежных реквизитов получателя платежа* — чтобы купить булочку, Джон должен указать в своем приложении адрес кафе и сумму платежа. Вводить эту информацию вручную трудно, и есть риск ошибиться, поэтому было бы неплохо, если бы Джон мог отсканировать реквизиты с помощью своей камеры.
- *Выполнение платежа* — приложение должно поддерживать возможность отправки электронного письма Лизе с платежной информацией и цифровой подписью.
- *Слежение за доступными средствами* — пользователи должны знать, сколько булочек они могут купить. Приложение должно отображать общее количество жетонов, имеющихся у пользователя.
- *Резервное копирование закрытых ключей* — когда закрытые ключи создаются в приложении, они существуют только в приложении. Если мобильный телефон потеряется или сломается, закрытые ключи пропадут. Вы уже знаете, что происходит при потере ключей, не так ли? Чтобы ключи не пропали, нужно хранить их резервные копии.

Группа разработчиков создала начальную версию приложения и назвала его кошельком. Термин *кошелек* не точен, потому что приложение не хранит денег. Оно хранит ключи, с помощью которых можно тратить деньги. Фактические деньги хранятся в электронной таблице. Приложение больше похоже на физическую связку ключей, но термин *кошелек* широко используется в мире Биткоин для обозначения программ, хранящих закрытые ключи, поэтому мы последуем за сложившейся традицией и двинемся дальше. Рассмотрим особенности этого кошелька.

Снова предположим, что Джон решил купить булочку в кафе (рис. 4.2). Джон и кафе используют это новое приложение.

Теперь процесс оплаты будет включать следующие шаги:

❶ Сотрудник кафе запускает кошелек и выбирает функцию создания нового адреса и запроса 10 CT по этому адресу. Этот новый адрес и сумма отображаются на экране в виде QR-кода. QR-код содержит всю информацию, необходимую для выполнения платежа, поэтому Джону не нужно вводить ее вручную.

QR-КОДЫ

QR-коды (QR, Quick Response) — это форма представления текстовой информации, пригодной для сканирования. Этот QR-код содержит текст «Привет!»:

❷ Джон направляет камеру своего телефона на QR-код, сканирует *URI платежа* (Uniform Resource Identifier — унифицированный идентификатор ресурса — стандартный способ идентификации чего бы то ни было; примером URI может служить веб-адрес URL):

```
ct:19UzNFW4Fq8wm8mtWmoPZAzE3tcB9tZVtN?amount=10
```

Он сообщает телефону Джона, что тот должен запустить кошелек с жетонами на булочки (`ct:`) и перевести 10 (`amount=10`) жетонов на адрес `19UzNFW4Fq8wm8mtWmoPZAzE3tcB9tZVtN`.

❸ Кошелек Джона отображает информацию о предстоящем платеже, и Джон, проверив ее, нажимает OK.

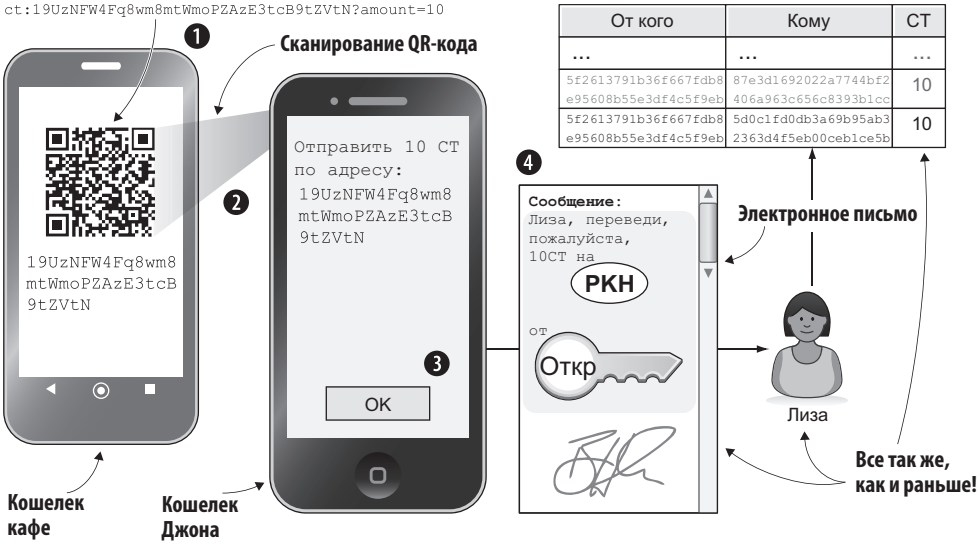


Рис. 4.2. Джон покупает булочки с помощью приложения кошелька. Сотрудник кафе генерирует ключ и показывает Джону изображение QR-кода с информацией о платеже. Джон сканирует эту информацию и касается кнопки OK, подтверждая платеж. После этого кошелек Джона посылает электронное письмо Лизе

❹ Кошелек Джона создает письмо Лизе, которое выглядит так же, как и раньше, автоматически выбирает адрес для отправки и подписывает сообщение правильным закрытым ключом. С точки зрения Лизы ничего не изменилось. Она проверяет и выполняет платеж точно так же, как и раньше.

BIP21

Предложения по улучшению Биткоин (Bitcoin Improvement Proposal, BIP) используются для передачи идей между разработчиками. Какие-то предложения BIP внедряются в программное обеспечение Биткоин, какие-то нет. Все предложения BIP перечислены на веб-ресурсе 9 (см. приложение B).

В Биткоин было реализовано предложение BIP21, касающееся способа передачи информации о платеже из одного кошелька в другой с использованием URI. Идентификаторы URI в Биткоин начинаются с префикса `bitcoin:` вместо `ct:`.

Рассмотрим детальнее, что делает кошелек Джона на шаге 4 (рис. 4.3). Фактически он выполняет те же действия, которые выполняли пользователи вручную в более ранних примерах.

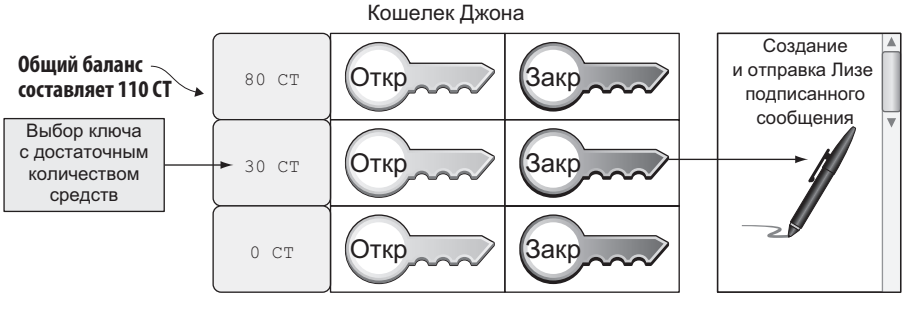


Рис. 4.3. Джон просто нажимает кнопку OK в своем кошельке, чтобы подтвердить платеж. Все остальное кошелек сделает автоматически. Он выберет ключ с достаточным количеством средств, создаст и отправит Лизе подписанное сообщение

Обратите внимание, что кошелек управляет тремя парами ключей: на двух из них имеются средства, а на одной нет. Благодаря этому новому кошельку пользователи могут иметь столько адресов, сколько захотят, что хорошо для конфиденциальности. Кошелек будет следить за ними автоматически.

Кошелек кафе, так же как кошелек Джона, будет периодически проверять появление новых записей в электронной таблице, касающихся любых ключей в кошельке, присутствующих в полях «Кому», «От кого» или в обоих (рис. 4.4).

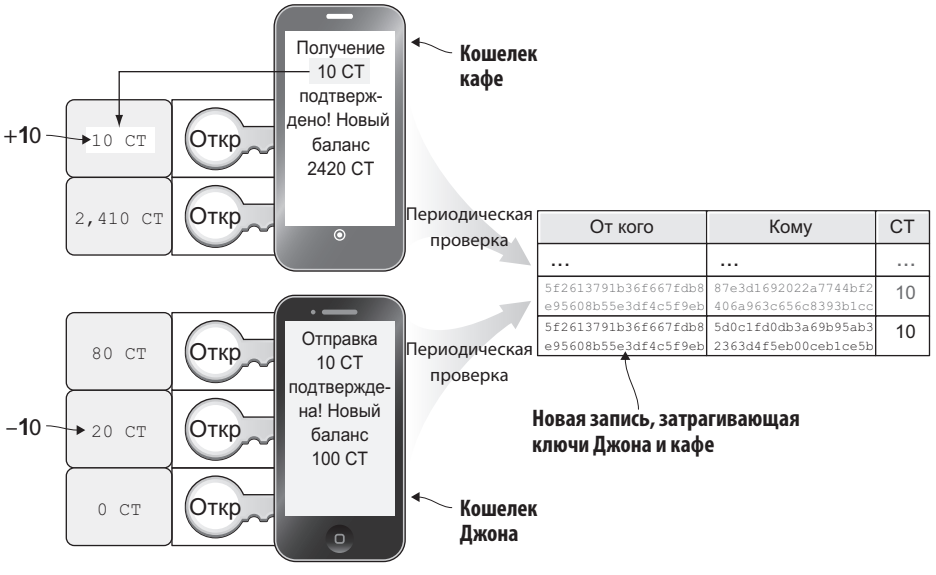


Рис. 4.4. Кошельки Джона и кафе проверяют содержимое электронной таблицы каждые несколько секунд. Если появится новый входящий или исходящий платеж, кошелек обновит баланс соответствующих ключей и уведомит пользователя

Джон знает, что перевел деньги, еще до того, как Лиза подтвердит платеж в электронной таблице, но его кошелек не обновит баланс до подтверждения. Почему? Потому что Лиза может не подтвердить платеж. Возможно, сообщение с информацией о платеже было повреждено во время передачи или электронное письмо оказалось у Лизы в папке со спамом и она не видит его.

Если кошелек обновит баланс, не дожидаясь появления подтверждения в электронной таблице, он может дать Джону ложную информацию. С другой стороны, кошелек может сообщить Джону, что платеж ожидает подтверждения.

НЕПОДТВЕРЖДЕННЫЕ ТРАНЗАКЦИИ

«Неподтвержденная» означает, что транзакция создана и отправлена в сеть Биткоин, но пока не стала частью цепочки блоков — блокчейна. Не следует доверять платежу, пока он не будет добавлен в блокчейн. То же относится и к платежам жетонами на булочки — не доверяйте платежам, которых нет в таблице.

Резервное копирование закрытых ключей

Разработчики реализовали в кошельке возможность резервного копирования закрытых ключей. Суть ее заключается в следующем: кошелек создает текстовый файл со всеми закрытыми ключами и посылает его на адрес электронной почты, указанный пользователем.

Представьте, что Джон решил создать резервную копию своих закрытых ключей. Кошелек собирает все когда-либо созданные ключи и записывает их в текстовый файл (рис. 4.5).

ЗАЧЕМ СОЗДАВАТЬ РЕЗЕРВНЫЕ КОПИИ?

Ваши ключи хранят ваши деньги. Потеряв ключи, вы потеряете деньги. Резервное копирование *не* является обязательным. Но определенно следует предпринять активные шаги, чтобы сохранить ключи; иначе рано или поздно вы потеряете деньги.

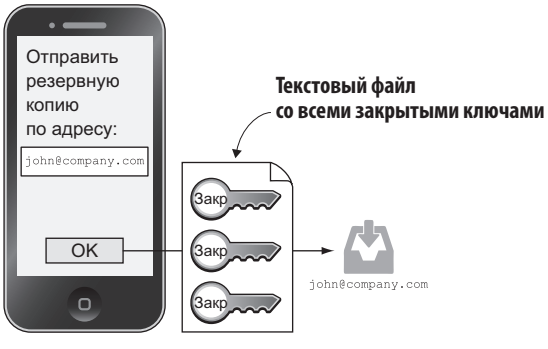


Рис. 4.5. Джон создает резервную копию своих закрытых ключей. Они записываются в текстовый файл, который затем отправляется на адрес электронной почты

Текстовый файл отправляется на адрес электронной почты Джона. Вы видите какие-либо проблемы с этим? Да, самая большая проблема в том, что ключи покинули конфиденциальную область приложения кошелька и отправляются на дикую природу. Любой имеющий доступ к серверу электронной почты или другим системам, вовлеченным в процесс пересылки, сможет получить закрытые ключи без ведома Джона.

ПРОБЛЕМЫ

- Риск кражи
- Частое резервное копирование утомляет

Но есть еще одна проблема. Если Джон создаст новый адрес после резервного копирования, этот новый адрес не сохранится в резервной копии. Джон должен создать еще одну резервную копию, включающую новый ключ. Для каждого нового

ключа он должен создавать новую резервную копию. В результате резервное копирование превращается в утомительный процесс для пользователя.

Рассмотрим несколько простых решений этих двух проблем:

1. Автоматическая отправка резервных копий после создания каждого адреса. Это увеличивает риск кражи, потому что резервные копии пересылаются чаще.
2. Заранее создать 100 адресов и создать их резервную копию. Эту процедуру можно повторить после исчерпания первой сотни адресов. Это решение также увеличивает риск кражи, но не так сильно, как решение 1.
3. Зашифровать резервную копию паролем. Это обезопасит от кражи ключей в резервной копии.

Комбинация решений 2 и 3 выглядит хорошей стратегией; в этом случае резервные копии создаются реже и они защищены надежным паролем.

Процесс копирования в этом случае напоминает предыдущий, но на этот раз Джон вводит пароль, который используется для шифрования закрытых ключей (рис. 4.6). Если Джон потеряет телефон, ему понадобятся лишь пароль и файл с резервной копией, чтобы восстановить свои закрытые ключи.

Рис. 4.6. Джон создает резервную копию своих закрытых ключей. Они записываются в файл, который затем шифруется паролем, который Джон вводит на своем телефоне

Если Джон потеряет свой телефон, он сможет установить приложение кошелька на другой телефон, переслать файл с резервной копией в приложение и ввести свой пароль. После этого файл будет расшифрован и ключи из него добавятся в приложение кошелька.

Несколько слов о надежных паролях

Надежность пароля измеряется *энтропией*. Чем выше энтропия, тем сложнее угадать пароль. Слово *энтропия*, используемое в сфере информационной безопасности, заимствовано из термодинамики и означает «беспорядок» или «неопределенность». Предположим, вы создали пароль длинной в 8 символов из следующих 64 символов:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

Каждый символ в таком пароле будет представлять 6 бит энтропии, потому что имеется $64 = 2^6$ возможных символов. Если выбрать 8 символов случайным образом (по-настоящему случайным!), скажем, `E3NrkbA7`, такой пароль будет иметь $6 \times 8 = 48$ бит энтропии. По надежности он эквивалентен 48 броскам монеты.

Предположим теперь, что вы выбираете случайные слова из словаря, содержащего $2^{11} = 2048$ слов. Сколько слов нужно использовать, чтобы преодолеть 48-битную энтропию вашего 8-символьного пароля? Четырех слов будет недостаточно, потому что $4 \times 11 = 44$ бита энтропии. Но пять слов соответствуют 55 битам энтропии, которая превосходит энтропию пароля.

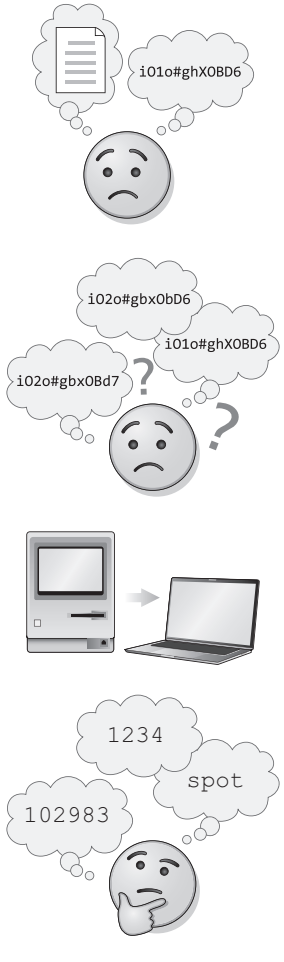
Истинная энтропия пароля также зависит от того, что злоумышленник знает о пароле. Например, допустим, что злоумышленник, пусть это будет Мэллори, украла зашифрованный файл Джона с резервной копией и пытается взломать его методом перебора. Атака *методом перебора* означает, что злоумышленник многократно пытается применить разные пароли, пока не найдет правильный. Если Мэллори знает, что длина пароля ровна 8, а символы выбираются из 64 упомянутых символов, энтропия пароля составит 48 бит. Если она узнает, что второй символ — `3`, энтропия уменьшится до $6 \times 7 = 42$ бит. С другой стороны, если Мэллори не знает длину пароля, ей будет сложнее, то есть энтропия будет выше.

Эти рассуждения верны, только если пароль действительно выбирался случайным образом. Если Джон выбирал пароль с некоторым умыслом, на-

пример `j0Hn4321`, его энтропия резко уменьшается. Типичные программы, реализующие атаку методом перебора, сначала пробуют применить множество известных слов и имен в разных вариациях и только потом пытаются генерировать более «случайные» пароли. Джон (John) — это распространенное имя, поэтому злоумышленник попробует применить разные варианты этого имени, а также множество других имен и слов. Например:

```
butter122 ... waLk129 ... go0die muh4mm@d
john John JOhn JOHn JOHN j0hn j0Hn
jOhn jOHn jOHN ... john1 ...
... john12 J0hn12 ... j0Hn321 ...
j0Hn4321
```

В яблочко! Предположим, что есть 1 000 000 распространенных слов и имен, и каждое слово может иметь в среднем 100 000 вариаций. При таких начальных условиях потребуется проверить до 100 миллиардов разных паролей, что соответствует примерно 37 битам энтропии. Чтобы произвести 100 миллиардов попыток, высокопроизводительному настольному компьютеру потребуется всего несколько дней. Для простоты допустим, что это займет один день. Если Джон использует по-настоящему случайный пароль, энтропия для атакующего составит около 48 бит. Взлом такого пароля займет около 2000 дней, или около 5,5 года.



Проблемы резервных копий, защищенных паролем

Шифрование резервных копий паролями — неплохое решение, но такой подход вводит новые проблемы:

- *Для поддержания безопасности требуется больше составляющих* — теперь Джон должен иметь два компонента: файл резервной копии и пароль. В первой версии ему нужен был только файл резервной копии.
- *Забытый пароль* — пароли, которые используются редко, такие как пароли резервных копий, в конечном итоге забываются. Эту проблему можно нивелировать, записывая пароли на бу-

маге и складывая их в надежном месте. Пароли также можно хранить с помощью программного обеспечения для управления паролями, например LastPass или KeePass.

- *Развитие технологий* — со временем создается новое, более совершенное аппаратное и программное обеспечение, которое ускоряет процесс взлома паролей. Если еще пять лет назад 8-символьный пароль считался достаточно надежным, то сейчас этого недостаточно. Энтропия паролей должна увеличиваться по мере совершенствования технологий. Можно, конечно, повторно шифровать файлы резервных копий каждые два года, используя более надежный пароль, но это сложный процесс, который под силу немногим пользователям.
- *Сложность в достижении случайности* — придумывать случайные пароли действительно сложно. Когда приложение запрашивает у Джона пароль, он должен тут же придумать его. У него нет времени, чтобы подбросить монету 48 раз и получить хороший пароль. Скорее всего, он придумает что-то с гораздо меньшей энтропией. Один из способов решить эту проблему — заставить кошелек сгенерировать пароль и предложить его Джону. Но запомнить этот пароль почти наверняка будет сложнее, чем пароль, придуманный вами самими, что увеличит вероятность забыть его.

Похоже, нам не удалось найти хорошего решения для резервного копирования. Но не будем соглашаться на имеющееся и недостаточно хорошее решение — есть варианты получше.

Иерархически детерминированные кошельки

Один из ярких разработчиков в компании, криптограф по образованию, предложила новый способ создания ключей, улучшающий ситуацию с резервным копированием и добавляющий в кошельки совершенно новые возможности.

Она заметила, что если все закрытые ключи в кошельке сгенерированы из одного случайного числа, называемого начальным числом, или сидом[1], случайной последовательности (random seed), весь бумажник

BIP32

В этом разделе описывается стандарт под названием BIP32, который широко используется разными программными кошельками в Биткоин. Стандарты BIP доступны на веб-ресурсе 9 (см. приложение B).

[1] От *англ.* seed — семя. — *Примеч. ред.*

можно сохранить, записав это число на листе бумаги и спрятав его в надежном месте (рис. 4.7).

Рис. 4.7. Сохранение резервной копии начального числа. Это лучший способ резервного копирования

Она пообщалась с другими криптографами, и вместе они выбрали стратегию. Они решили создать HD-кошелек. Ключи в таком кошельке организованы в виде дерева, в котором один ключ является корнем дерева, и этот корень может иметь любое количество дочерних ключей. Каждый дочерний ключ, в свою очередь, может иметь свои дочерние элементы и т. д.

Предположим, что Рита решила организовать свои ключи по назначению и создать пять ключей для покупок в кафе и еще три ключа для использования в качестве сберегательного счета. На рис. 4.8 показано, как могли бы быть организованы ее ключи.

BIP44

BIP44, Multi-Account Hierarchy for Deterministic Wallets (Многоуровневая иерархия для детерминированных кошельков), описывает, какие ветви дерева и для каких целей используются. А пока воспользуемся организацией ключей, выбранной Ритой.

Ключи организованы в виде дерева, но это дерево перевернуто с ног на голову, потому что именно так компьютерные гики обычно рисуют свои деревья. В любом случае корневой ключ дерева (вверху) называется *главным закрытым ключом*. Из этого ключа порождаются все остальные ключи. Главный закрытый ключ имеет два *дочерних ключа*: один представляет счет для покупок (слева на рис. 4.8), а другой — сберегательный счет (справа). У каждого из этих дочерних ключей есть свои дочерние ключи.

Ключ для покупок имеет пять дочерних ключей, а ключ сберегательного счета — три дочерних ключа. Эти восемь дочерних ключей не имеют своих потомков, поэтому их называют *листьями* дерева. Листья — это закрытые ключи, которые Рита использует для хранения жетонов на булочки, поэтому для каждого из этих восьми закрытых ключей генерируется адрес.

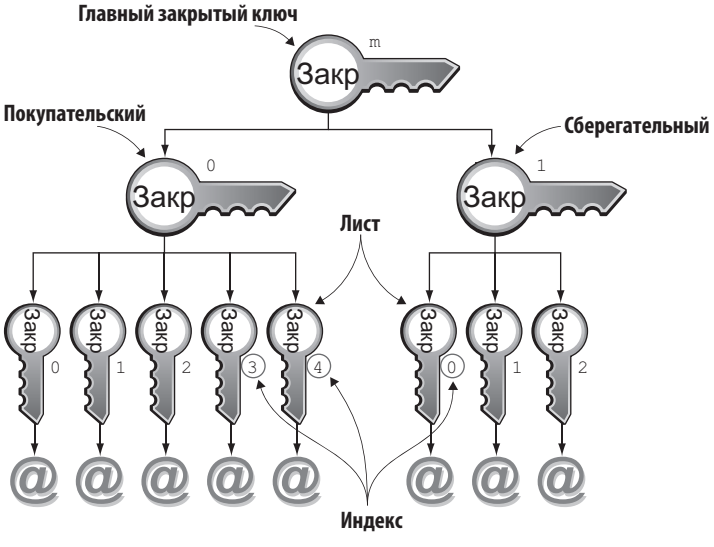


Рис. 4.8. Рита создает два счета и пять адресов для покупательского счета и три для сберегательного

Обратите внимание на нумерацию ключей в дереве. Каждый набор дочерних ключей нумеруется, начиная с 0. Благодаря этому каждый ключ получает уникальный идентификатор. Например, первый ключ сберегательного счета с *индексом* 0 обозначается как `m/1/0`, где `m` является специальным обозначением главного закрытого ключа.

ИНДЕКСЫ

Программисты часто используют термин *индекс* для обозначения позиции в списке. Обычно счет начинается с нуля, то есть первый элемент в списке имеет индекс 0, второй элемент имеет индекс 1 и т. д. В этой книге мы будем использовать индексы, начинающие счет с 0.

Как формируется древовидная структура? Давайте внимательно рассмотрим создание некоторых частей дерева. При создании дерева выполняются три важных процесса, как показано на рис. 4.9:

1. Генерируется случайное 128-битное число. Это начальное число (семя), из которого будет расти дерево.
2. Из начального числа генерируется *главный расширенный закрытый ключ*.
3. Из главного расширенного закрытого ключа генерируются *дочерние расширенные закрытые ключи*.

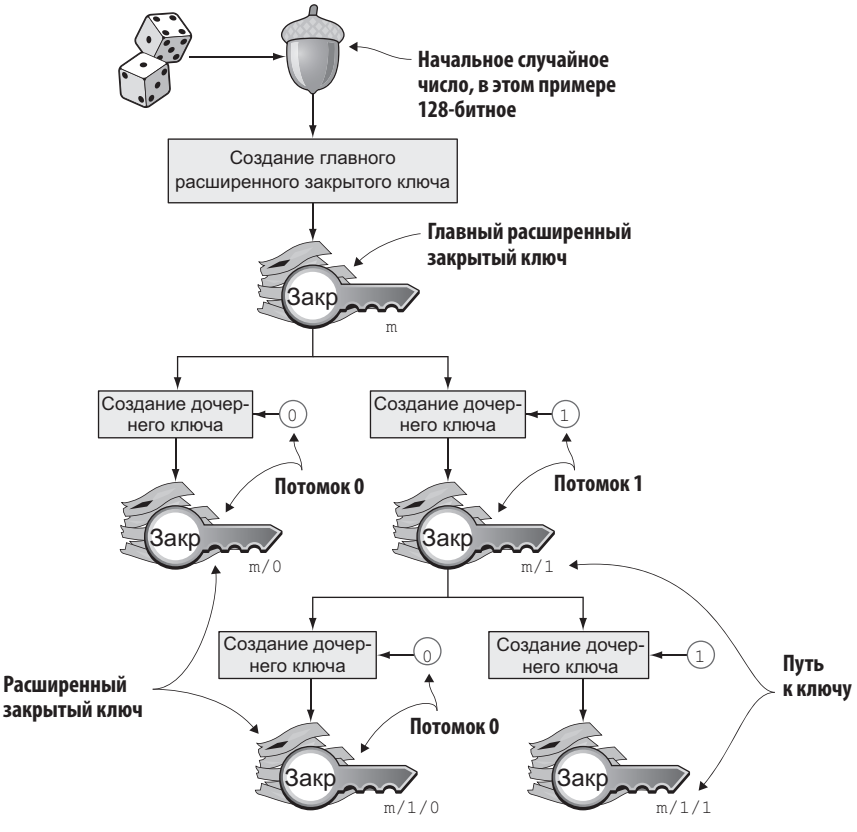
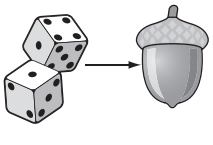


Рис. 4.9. Создание первых двух из трех сберегательных ключей Риты. На основе случайного начального числа генерируется главный расширенный закрытый ключ, который затем используется для создания дочерних расширенных закрытых ключей

Расширенный закрытый ключ (extended private key, xprv) состоит из двух компонентов: закрытого ключа и цепного кода (chain code, см. рис. 4.10).

Рис. 4.10. xprv состоит из закрытого ключа и цепного кода

Новый закрытый ключ ничем не отличается от закрытого ключа старого типа, сгенерированного непосредственно генератором случайных чисел. Его также можно использовать для создания открытого ключа и адреса. Обычно адреса создаются только для листьев, но их можно создавать также для промежуточных ключей. Другой компонент xprv — цепной код. Цепной код — это крайние правые 256 бит 512-битного хеша, что объясняет вид значка на рисунке. Чуть ниже я покажу, как создается этот хеш. Целью цепного кода является обеспечение энтропии при создании дочернего xprv. Главный xprv не отличается от других xprv, но мы дали ему особое название (главный), потому что он является прародителем всех ключей в дереве, но создается иным способом.



Начальное случайное число создается на первом шаге точно так же, как создавались закрытые ключи в главе 2. В этом примере генерируется 128 бит случайных данных, но точно так же можно сгенерировать 256 бит, в зависимости от желаемого уровня безопасности, однако для большинства пользователей вполне достаточно 128 бит. Далее вы увидите, как выбор размера начального числа влияет на процесс резервного копирования; более длинное начальное число дольше записывать на листе бумаги во время резервного копирования. Мы вернемся к этой теме в разделе «Назад к резервному копированию».

Описание шагов 2 и 3 заслуживает отдельных подразделов, которые следуют ниже.

Создание главного расширенного закрытого ключа

Давайте посмотрим, как создается главный xprv (рис. 4.11).

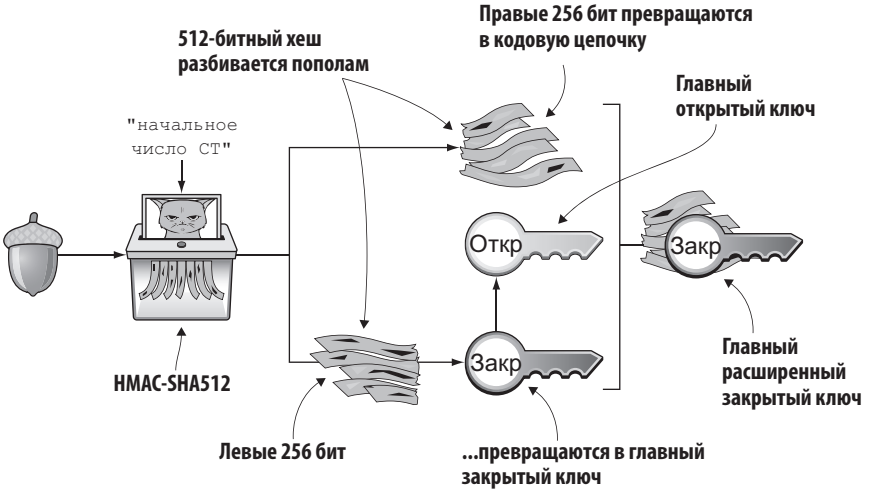
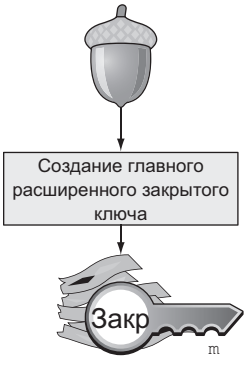


Рис. 4.11. Создание главного xprv Риты. Начальное случайное число хешируется с использованием алгоритма HMAC-SHA512. Получившийся 512-битный хеш разбивается на левые 256 бит, которые превращаются в главный закрытый ключ, и правые 256 бит, которые превращаются в цепной код

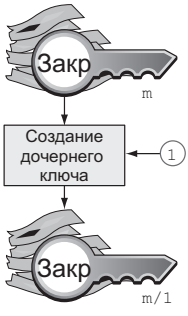


Чтобы создать главный закрытый ключ, начальное число хешируется с использованием алгоритма HMAC-SHA512 (HMAC — сокращенно Hash Based Message Authentication Code, хеш-код аутентификации сообщений), которое генерирует 512-битный хеш. HMAC-SHA512 — это специальная криптографическая хеш-функция, которая помимо входного числа принимает дополнительный ключ. С точки зрения пользователя алгоритм HMAC-SHA512 можно считать обычной криптографической хеш-функцией, но с несколькими входами. Значение хеш-функции делится на левые и правые 256 бит. Левые 256 бит становятся главным закрытым ключом, который является самым обычным закрытым ключом; он называется *главным*, потому что все остальные закрытые ключи получаются из этого единственного закрытого ключа (и цепного кода). Правые 256 бит становятся *цепным кодом*, который используется на следующем шаге для получения потомков главного xprv.

«НАЧАЛЬНОЕ ЧИСЛО CT»?

Хеш-функции HMAC требуется два входных значения: число для хеширования и ключ. Для создания главного xprv ключ не требуется, потому что вся необходимая энтропия заключена в начальном числе. На рис. 4.11 мы вводим «начальное число CT», чтобы *что-то* передать хеш-функции HMAC. Ключ нам потребуется потом, когда будем создавать дочерние ключи на основе главного xprv.

Создание дочернего расширенного закрытого ключа



Мы только что создали главный xprv Риты. Теперь создадим дочерний xprv, объединяющий три ее сберегательных ключа. Прямые потомки xprv могут генерироваться в любом порядке. Для начала получим ключ сберегательного счета, `m/1`. Процесс получения дочернего xprv из родительского выглядит следующим образом (рис. 4.12):

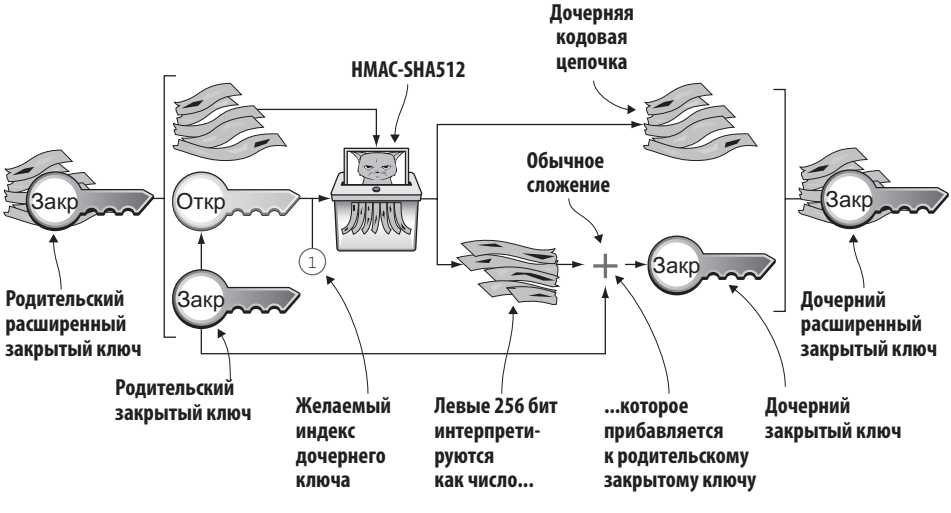


Рис. 4.12. Создание дочернего xprv из родительского. Сначала вместе хешируются открытый родительский ключ, цепной код и требуемый индекс. Затем закрытый родительский ключ складывается с левой половиной хеша и сумма становится закрытым дочерним ключом. Правая половина становится дочерним цепным кодом

1. Желаемый индекс добавляется в конец родительского открытого ключа.
2. Открытый ключ с индексом подаются на вход HMAC-SHA512. Родительский цепной код выступает в роли источника энтропии для хеш-функции. Для простоты можно считать, что все три элемента данных хешируются вместе.
3. 512-битный хеш разбивается пополам:
   - левые 256 бит складываются обычным образом (по модулю 2^{256}) с родительским закрытым ключом, и сумма становится дочерним закрытым ключом;
   - правые 256 бит становятся дочерним цепным кодом.
4. Дочерний закрытый ключ и цепной код вместе образуют дочерний xprv.

Этот процесс используется для создания всех дочерних и внучатых xprv, пока не будут получены все ключи, которые Рита хотела бы хранить в своем кошельке.

Вы можете спросить, зачем нужна эта операция сложения — почему бы просто не использовать оставшиеся 256 бит в качестве дочернего закрытого ключа? 512-битный хеш рассчитывается на основе открытого ключа с цепным кодом, которые вместе называются *расширенным открытым ключом* (extended public key, xpub), и индекса. Далее вы увидите, как использовать xpub в слабо защищенных окружениях, таких как веб-серверы, для создания соответствующего дерева *открытых* ключей. Сложение родительского закрытого ключа с левыми 256 битами необходимо, чтобы никто не смог сгенерировать дочерние закрытые ключи по имеющемуся xpub.

На чем мы остановились?

Давайте вспомним, почему мы здесь оказались: чтобы создать приложение кошелька, которое облегчит жизнь конечным пользователям (рис. 4.13).

Основные задачи кошелька:

- управление закрытыми ключами;
- создание новых адресов;
- отправка реквизитов платежа от получателя плательщику;

- выполнение платежей;
- слежение за имеющимися средствами;
- резервное копирование закрытых ключей.

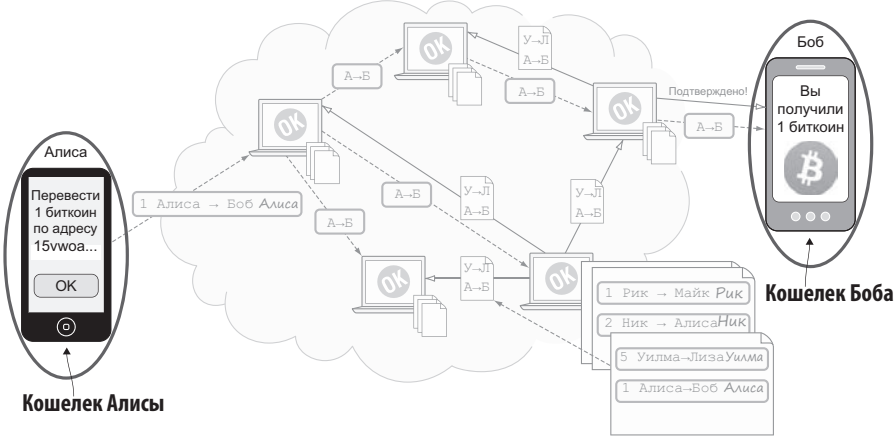


Рис. 4.13. Мы работаем над созданием кошелька, удобного для пользователей

Мы исследовали первые пять пунктов, но пока не закончили с резервным копированием. Мы только что познакомились с процессом создания расширенных закрытых ключей (xprv), лежащим в основе более удачной процедуры резервного копирования.

Назад к резервному копированию

Нужен простой и безопасный способ резервного копирования закрытых ключей. Мы создали HD-кошелек, который может сгенерировать любое количество закрытых ключей из одного начального случайного числа. Какой минимальный объем информации должна сохранить Рита в резервной копии, чтобы после потери кошелька она смогла восстановить все свои ключи? Верно: достаточно сохранить начальное случайное число (и древовидную структуру, см. при-

А ПУТИ К КЛЮЧАМ?

Для восстановления ключей необходимо также запомнить пути к ним. В Биткоине эти пути стандартизированы в BIP44. Если кошелек использует этот стандарт, вы неявно знаете пути ключей.

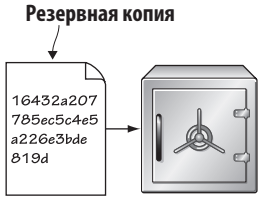
мечание). Пока ее начальное случайное число хранится в безопасности, она в любой момент сможет заново создать все свои ключи.

Допустим, что Рита выбрала начальное 128-битное (16-байтное) случайное число:

```
16432a207785ec5c4e5a226e3bde819d
```

Очевидно, что записать эти 32 шестнадцатеричные цифры на листе бумаги гораздо проще, чем восемь закрытых ключей. Но самая большая выгода в том, что Рита может записать это число один раз и запереть в сейфе. Пока этот лист бумаги в безопасности, ее кошелек защищен от случайной потери. Она может даже создавать новые ключи из того же начального числа, не обновляя резервную копию.

Но и это число трудно записать без ошибок. Что, если Рита допустила опечатку, а затем потеряет свой кошелек? Она не сможет восстановить ни один из своих ключей! Нам нужно нечто более простое, более приспособленное для людей.



Мнемонические предложения

Итак, начальное число представлено последовательностью битов. Например, начальное число Риты имеет длину 128 бит. Можно ли представить эту последовательность в форме, более удобной для запоминания? Оказывается, можно!

Кошелек Риты может отображать начальное число в виде последовательности из 12 английских слов, называемой *мнемоническим предложением*.

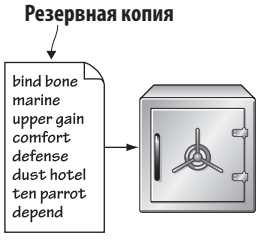
BIP39

Большинство кошельков в Биткоин используют мнемонические предложения для резервного копирования. Это закреплено в стандарте BIP39. До этого в кошельках обычно использовались файлы со всеми ключами, защищенные паролем, что было очень неудобно.

Число: `16432a207785ec5c4e5a226e3bde819d`

Мнемоника: `bind bone marine upper gain comfort defense dust hotel ten parrot depend`

Это мнемоническое предложение представляет начальное число в легко запоминаемой форме. Гораздо проще записать 12 слов, чем загадочный шестнадцатеричный код. Если Рита потеряет свой кошелек, она сможет установить приложение кошелька на другой телефон, восстановить начальное число по этим 12 словам и затем восстановить все свои закрытые ключи.



Преобразование начального числа в мнемоническое предложение

ВНИМАНИЕ

Далее рассматривается, как выполняется такое преобразование. Это действительно интересно, но если вы решите, что этот раздел слишком сложный, можете принять на веру предыдущий абзац и перейти к разделу «Расширенные открытые ключи».

Преобразование начинается со случайного начального числа, как показано на рис. 4.14. Это число хешируется с использованием алгоритма SHA256, и первые 4 бита хеша — в данном случае `0111` — добавляются в конец начального числа. Эти 4 бита действуют подобно контрольной сумме. Затем биты разбиваются на 12 групп по 11 бит, где каждая группа представляет число в диапазоне от 0 до 2047. С помощью одиннадцати бит можно представить $2^{11} = 2048$ разных значений, помните?

Затем выполняется поиск соответствий этим 12 числам в стандартизированном списке из 2048 слов, пронумерованных от 0 до 2047. Этот список, который можно найти в описании стандарта BIP39 на веб-ресурсе 9 (приложение B), содержит часто используемые английские слова. Результатом поиска соответствий для всех 12 чисел является мнемоническое предложение.

Предложение не имеет особого смысла — это 12 случайных слов, точно так же как начальное случайное число содержит 32 случайные шестнадцатеричные цифры.

Кошелек Риты показывает ей мнемоническое предложение, и она записывает 12 слов на листе бумаги. Она прячет лист в безопасном месте и продолжает спокойно жить.

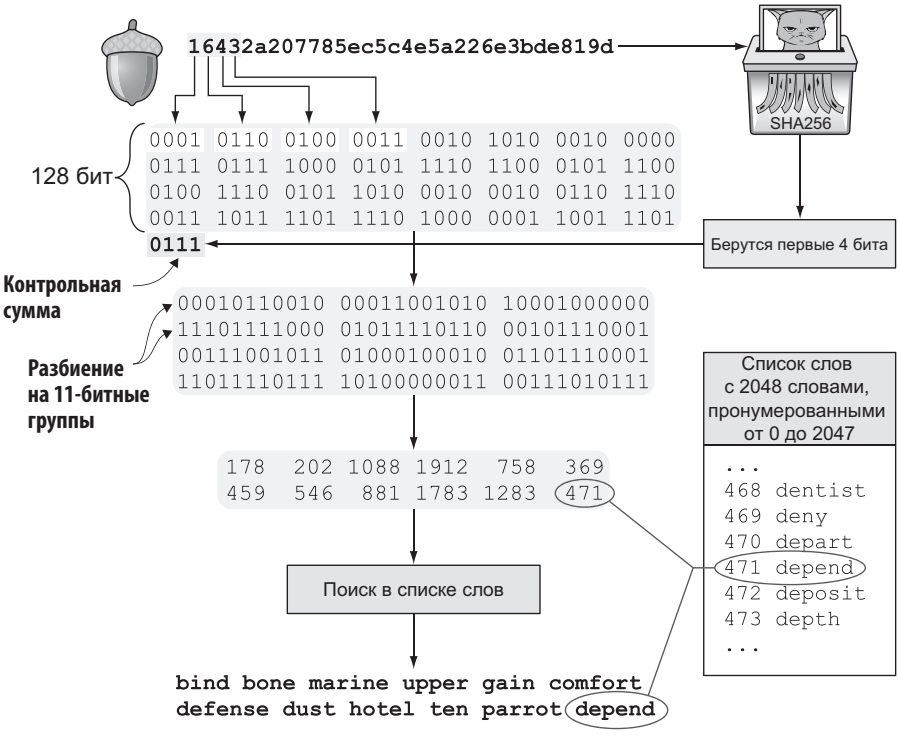


Рис. 4.14. Преобразование начального случайного числа в мнемоническое предложение с 12 словами. К числу добавляется контрольная сумма, и затем из списка с 2048 словами для каждой 11-битной группы выбирается слово с соответствующим порядковым номером

Обратное преобразование мнемонического предложения в начальное число

На следующий день во время морской прогулки Рита роняет свой телефон, и тот исчезает в морской пучине. Она потеряла свой кошелек! Но Риту это мало беспокоит. Она покупает новый телефон, устанавливает приложение кошелька и выбирает операцию восстановления ключей из резервной копии. В ответ кошелек предлагает ввести мнемоническое предложение. Она вводит

```
bind bone marine upper gain comfort
defense dust hotel ten parrot depend
```

Приложение кошелька выполняет обратное преобразование мнемонического предложения и из восстановленного начального случайного числа генерирует ключи, как показано на рис. 4.15.

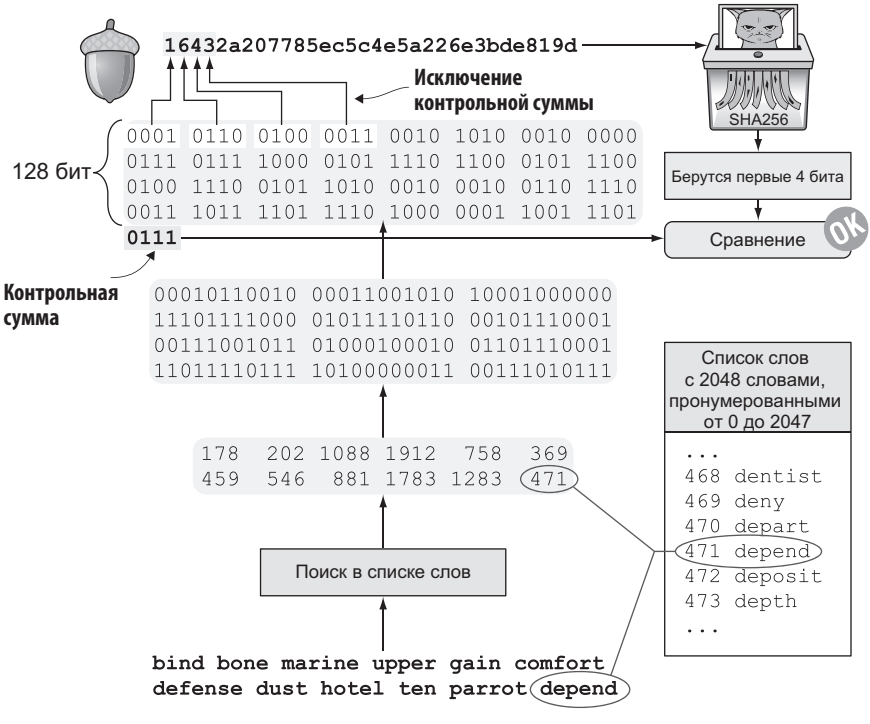


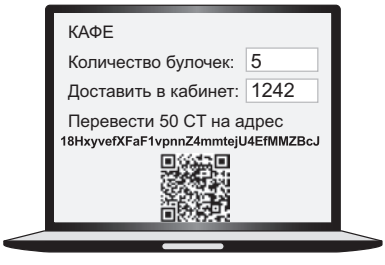
Рис. 4.15. Обратное преобразование мнемонического предложения в начальное случайное число

Процедура обратного преобразования использует 4-битную контрольную сумму, чтобы убедиться в правильности мнемонического предложения. Если Рита случайно запишет последнее слово как `deposit` вместо `depend`, проверка контрольной суммы *наверняка* потерпит неудачу. Если она введет `depends` вместо `depend`, проверка также завершится неудачей, потому что в списке нет такого слова.

В данном случае такая контрольная сумма — довольно слабая защита, потому что 4 бита дают всего 16 возможных контрольных сумм. Неправильно записанное мнемоническое предложение, в котором все слова присутствуют в списке слов, будет с вероятностью 1/16 принято как правильное. Такая степень защищенности от ошибок не выглядит достаточной. Но вероятность

написать такое предложение мала, потому что слова с ошибками должны присутствовать в списке слов. Это снижает риск восстановления ошибочного начального числа из неверного мнемонического предложения.

Расширенные открытые ключи



Рита создала свой кошелек из случайного 128-битного начального числа, которое она сохранила в виде мнемонического предложения из 12 слов. Ее кошелек может создать любое количество закрытых ключей из этого начального числа. Она может организовать их в разные «счета», как ей заблагорассудится. Это само по себе очень хорошо. Но HD-кошельки имеют еще одну интересную особенность: они способны создавать деревья открытых ключей и цепного кода вообще без использования закрытых ключей.

Предположим, что в кафе используется HD-кошелек и его владелец решил организовать продажу булочек через веб-сайт и их доставку в кабинеты работников.

По соображениям конфиденциальности веб-сервер должен иметь возможность предлагать новый адрес для каждой покупки, но где он получит эти адреса? Кафе может создать xprv для счета *онлайн-продаж* в своем HD-кошельке и разместить этот xprv на веб-сервере, как показано на рис. 4.16.

Теперь веб-сервер может создавать новые адреса по мере поступления заказов. Отлично! Но что, если Мэллори, наша злоумышленница, получит доступ к жесткому диску веб-сервера? Она сможет украсть все деньги с любого из адресов счета онлайн-продаж. Правда, при этом она не сможет украсть деньги с других адресов в дереве. Например, она не сможет вычислить ключи для *прямых продаж*, поскольку у нее нет доступа к главному xprv, который необходим, чтобы получить ключ счета для прямых продаж и всех его дочерних ключей.

Типичные веб-серверы часто подвергаются попыткам взлома, потому что доступны из любой точки мира. Хранение денег на веб-сервере наверняка повлечет много попыток взлома. Рано или поздно кому-то удастся получить доступ к жесткому диску веб-сервера и украсть xprv.

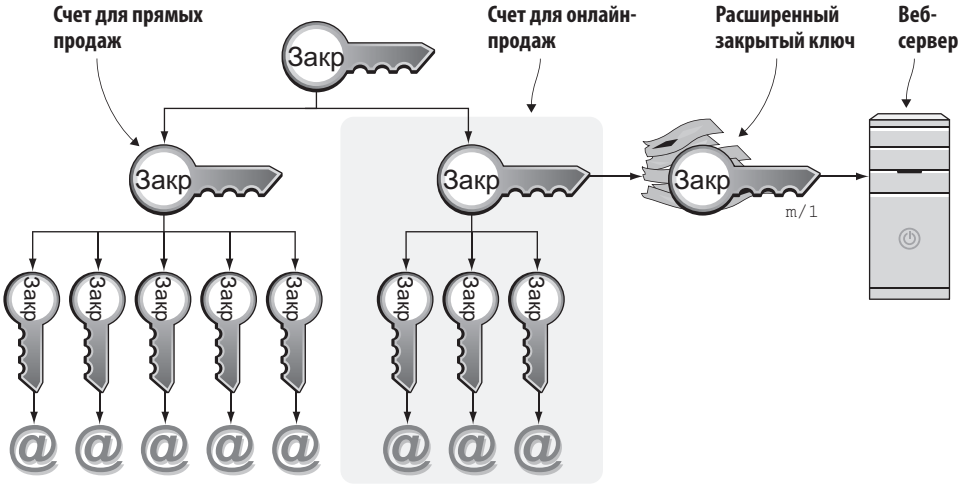


Рис. 4.16. Кафе копирует свой xprv для онлайн-продаж на веб-сервер

По этой причине владелец кафе хотел бы исключить использование любых закрытых ключей на веб-сервере. HD-кошелек позволяет это с помощью расширенных открытых ключей xpub (рис. 4.17).

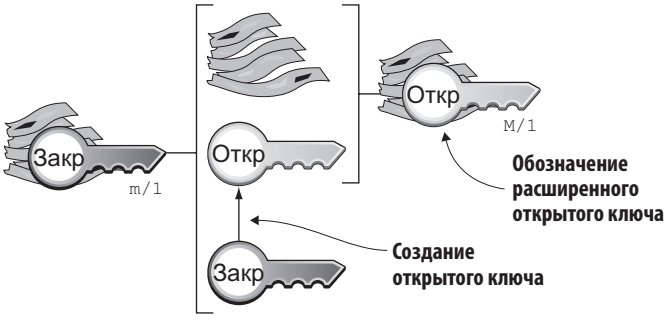


Рис. 4.17. Расширенный открытый ключ состоит из открытого ключа и цепного кода

Xpub напоминает xprv, но содержит открытый ключ и цепной код, тогда как xprv содержит закрытый ключ и цепной код. Xprv и xpub содержат один и тот же цепной код. Xpub можно создать из xprv, но нельзя создать xprv из xpub. Причина в том, что создание открытого ключа — однонаправленный процесс; открытый ключ можно получить из закрытого, но получить закрытый ключ из открытого невозможно.

Кафе помещает xpub `M/1` на веб-сервер. По соглашению для обозначения пути к xpub используется символ `M`, а для обозначения пути к xprv — символ `m`. `M/1` и `m/1` имеют одинаковый цепной код, но `M/1` включает только открытый ключ. Из главного xpub можно сгенерировать все дерево xpub (рис. 4.18), а значит, можно генерировать любые адреса без использования любых закрытых ключей. Вы сможете создавать адреса, но потратить деньги с этих адресов у вас не получится.

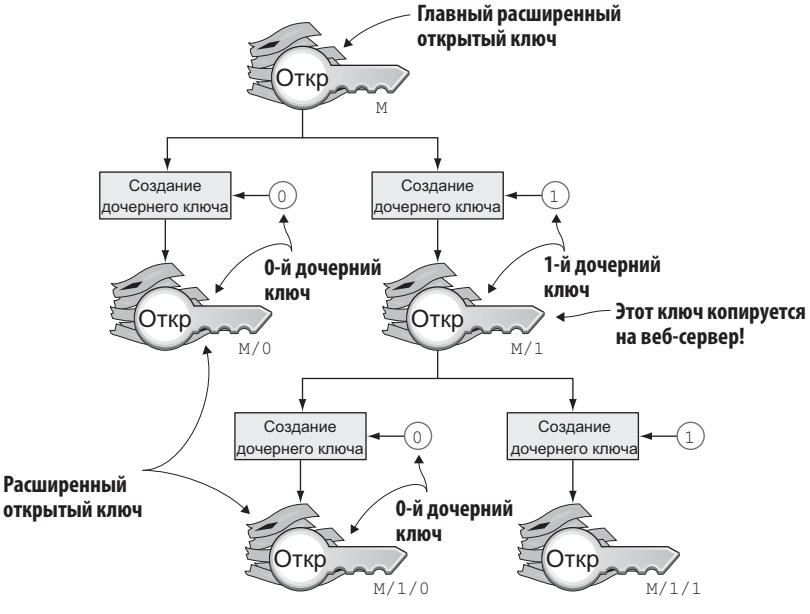
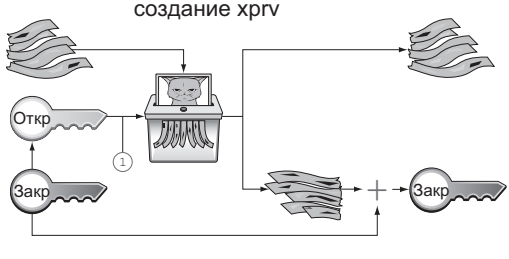


Рис. 4.18. Создание дерева xpub из главного xpub. Общий шаблон тот же, что и при создании xprv, но используется иная функция создания дочерних ключей

В целом процесс напоминает создание дерева xprv, отличаясь лишь отсутствием закрытых ключей. Однако, как показано на рис. 4.19, xpub генерируются не так, как xprv. Сравните эту процедуру с созданием xprv.



Сходство с процедурой создания xprv очевидно. Разница лишь в том, как используются левые 256 бит из 512-битного хеша. В схеме создания дочернего открытого ключа левые 256 бит используются в роли закрытого ключа, для

которого вычисляется промежуточный открытый ключ. Этот промежуточный открытый ключ затем складывается с родительским открытым ключом с помощью операции специального *сложения с открытым ключом*. В результате получается дочерний открытый ключ. Сравним создание дочернего открытого ключа с созданием дочернего закрытого ключа (рис. 4.20) после момента извлечения левых 256 бит из хеша HMAC-SHA256.

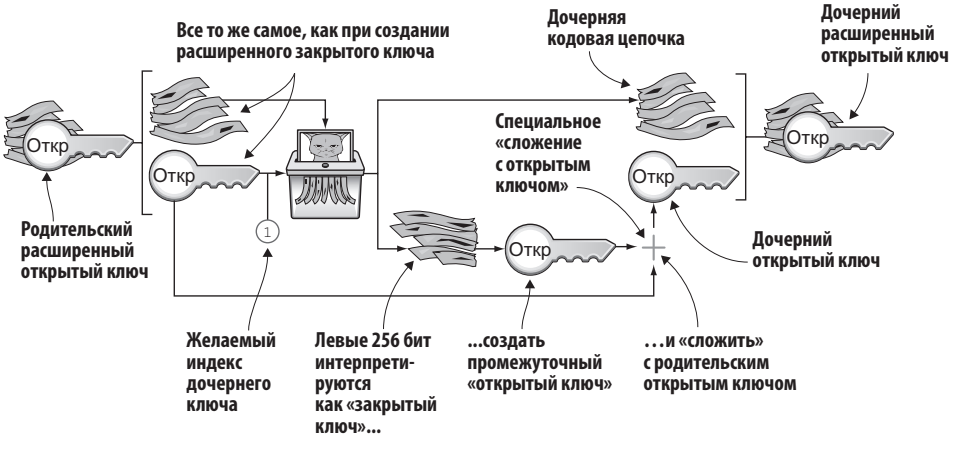


Рис. 4.19. Создание xpub. Сложение с закрытым ключом в процедуре создания xprv заменило «сложение» с открытым ключом

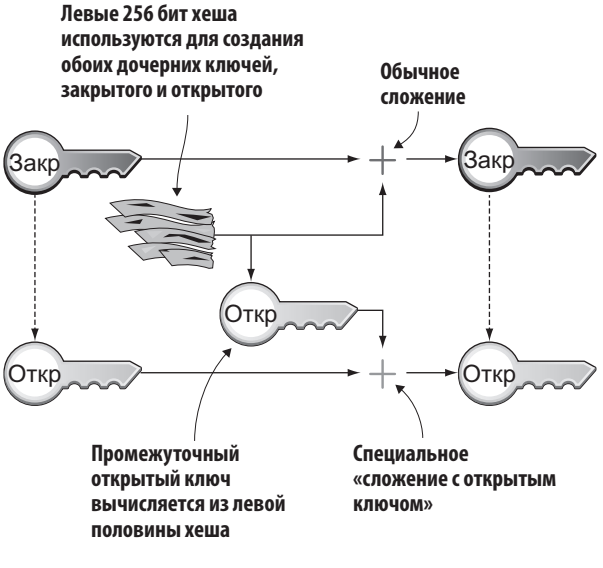


Рис. 4.20. Операция сложения в процедуре создания закрытого ключа соответствует операции сложения в процедуре создания открытого ключа. Сложение родительского закрытого ключа с некоторым значением дает в результате дочерний закрытый ключ. Сложение родительского открытого ключа с промежуточным открытым ключом, полученным из того же значения, дает в результате дочерний открытый ключ

В создании закрытого ключа используется обычное сложение: чтобы получить дочерний закрытый ключ, нужно 256-битное число сложить с родительским закрытым ключом. Но чтобы длина результата не превысила 256 бит, используется сложение *по модулю 2*256.

Сложение, используемое в создании дочернего открытого ключа, отличается от привычного для большинства людей (включая меня). Но давайте пока просто примем на веру, что это сложение работает. Мы подробно рассмотрим его в разделе «Математика открытого ключа».

Создание защищенных закрытых ключей

ВНИМАНИЕ

Это очень сложный раздел. Если у вас возникли трудности с пониманием того, как создаются xprv и xpub, советую пропустить этот раздел и перейти к «Математике открытого ключа». Сведения в этом разделе не нужны для понимания остальной части этой книги.

В этом разделе мы поговорим о том, как предотвратить возможные проблемы безопасности, связанные с обычной процедурой создания xprv.

Запуск онлайн-продаж в кафе начал приносить прибыль. Люди как сумасшедшие стали заказывать булочки! С созданием новых открытых ключей для заказов счет онлайн-продаж растет. Xpub для счета онлайн-продаж хранится на веб-сервере, а xprv присутствует только в кошельке кафе (и в мнемоническом предложении, запертом в сейфе).

Предположим, что каким-то образом Мэллори смогла украсть закрытый ключ `m/1/1`, на котором хранится всего 10 СТ. Такая кража может показаться безобидной, потому что на этом закрытом ключе так мало денег. Но на самом деле ситуация намного хуже. Если Мэллори удастся также заполучить xpub для счета онлайн-продаж, который хранится на веб-сервере, она сможет *вычислить xprv для счета онлайн-продаж*, как показано на рис. 4.21.

Помните, как функция создания xprv использовала обычное сложение для вычисления дочернего закрытого ключа из родительского?

`m/1` + левая половина хеша с индексом 1 = `m/1/1`

Это уравнение можно переписать так:

`m/1/1` – левая половина хеша с индексом 1 = `m/1`

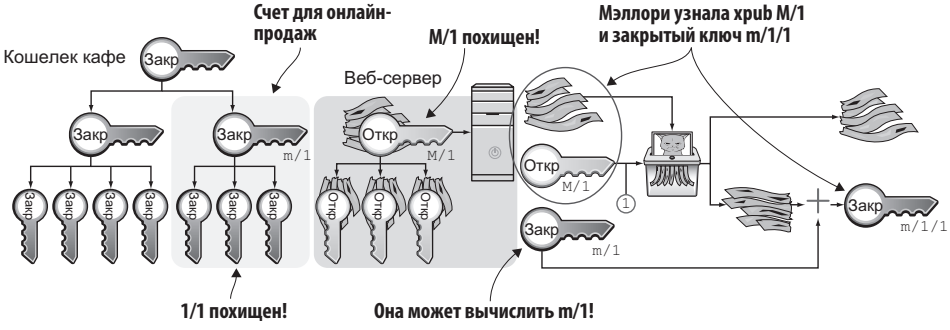


Рис. 4.21. Мэллори похитила закрытый ключ m/1/1 из кошелька кафе и родительский xpub с веб-сервера. Теперь она может украсть все деньги, имеющиеся на счете онлайн-продаж

У Мэллори есть все, что нужно, чтобы вычислить левую половину хеша для любого дочернего индекса `M/1`, который ей понравится, но она не знает, какой индекс имеет украденный ею закрытый ключ, поэтому начинает тестирование с индекса 0:

`m/1/1` – левая половина хеша с индексом 0 = закрытый ключ

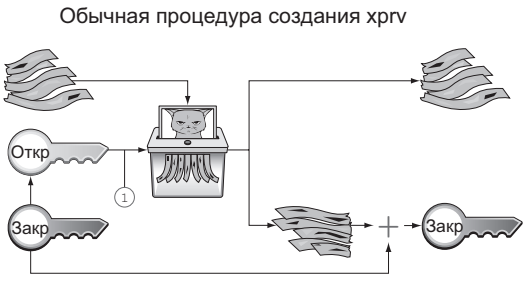
Она вычисляет открытый ключ для этого закрытого ключа и замечает, что он отличается от `M/1`, значит, 0 — ошибочный индекс. Затем она пробует индекс 1:

`m/1/1` – левая половина хеша с индексом 1 = другой закрытый ключ

Мэллори вычисляет другой закрытый ключ из открытого ключа `M/1` и на этот раз попадает точно в яблочко! Она получила закрытый ключ `m/1` счета для онлайн-продаж. Оба ключа, xprv и xpub, используют один и тот же цепной код, поэтому удалось вычислить xprv для `m/1`. Теперь она может вычислить все дерево закрытых ключей для счета онлайн-продаж и украсть все деньги. Это очень плохо.

А теперь представьте, что может случиться, если Мэллори завладеет главным xpub. Она сможет тем же способом, имея главный xpub и `m/1/1`, вычислить главный xprv и воссоздать все закрытые ключи для всех счетов во всем кошельке. Можно ли как-то предотвратить такой катастрофический сценарий? Да, если использовать *еще одну функцию создания ключей*! Эта новая функция называется функцией *создания защищенных xprv*.

Предположим, что в кафе решили предотвратить возможность вычисления главного xprv по известному главному xpub и закрытому ключу в счете онлайн-продаж. Для этого они могут сгенерировать xprv для счета онлайн-продаж, используя функцию создания защищенных xprv, как показано на рис. 4.22.



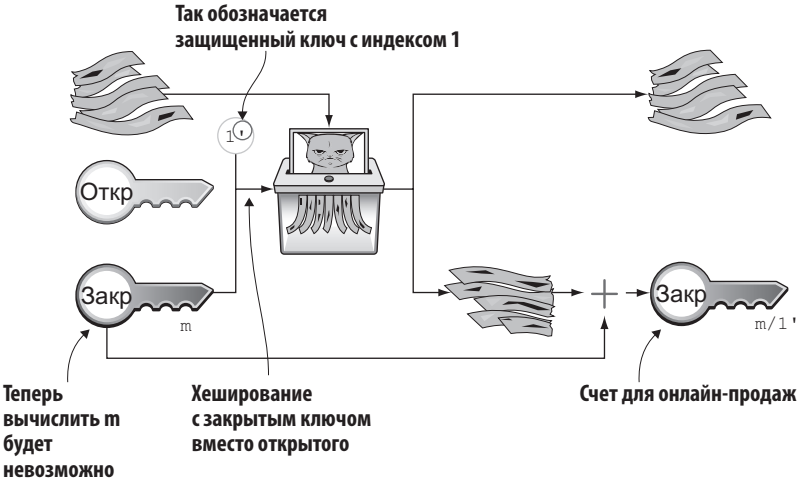


Рис. 4.22. Создание защищенного дочернего xprv для счета онлайн-продаж. На вход хеш-функции вместо открытого ключа подается родительский закрытый ключ

Апостроф в пути `m/1'` — это не опечатка: так обозначаются защищенные ключи. Разница между защищенным и незащищенным ключами состоит в том, что при создании защищенного ключа вместо открытого ключа хешируется *закрытый* ключ. В этом случае злоумышленник не сможет проделать трюк с вычитанием, потому что хеш получен из родительского закрытого ключа. Мэллори не сможет вычислить левую половину хеша для вычитания из дочернего закрытого ключа, потому что не имеет родительского закрытого ключа. Результат можно видеть на рис. 4.23.

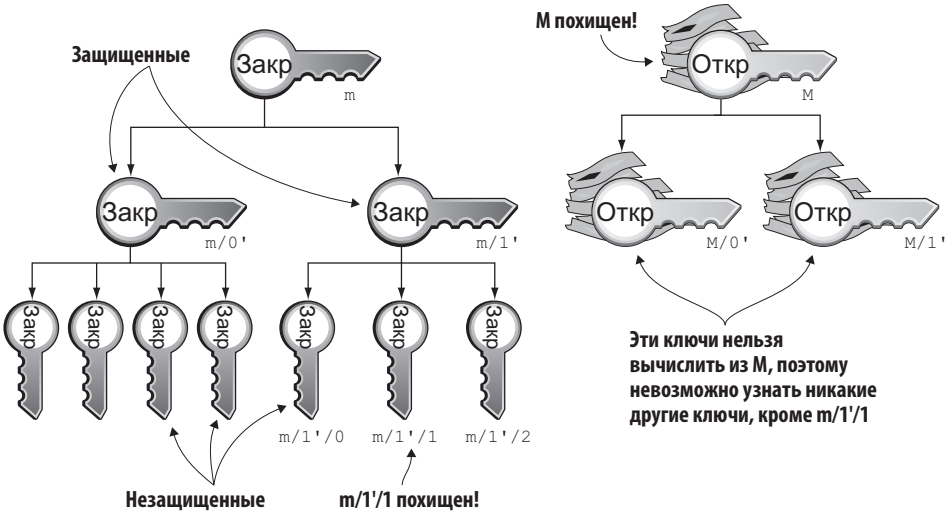


Рис. 4.23. Главный xpub не получится использовать для вычисления дочерних ключей, потому что m/0′ и m/1′ — это защищенные ключи

Это также означает, что нельзя создать защищенный дочерний xpub из родительского xpub. Чтобы создать защищенный ключ, открытый или закрытый, нужно иметь родительский xprv. Закрытые ключи, дочерние по отношению к `m/1'`, нельзя создать как защищенные, поскольку для этого потребуется, чтобы в кафе поместили закрытый ключ `m/1'` на веб-сервер онлайн-продаж, что небезопасно. Использование незащищенных ключей-листьев внутри счета онлайн-продаж делает кафе уязвимым для злоумышленника, укравшего `m/1'/1` и `M/1'`. Завладев этими ключами, он сможет украсть все средства, имеющиеся на счете. Защищенный xprv решает проблему воровства `M` и `m/1'/1`, но бессилен помочь в сценарии с похищением `M/1'` и `m/1'/1`.

Математика открытого ключа

В этом разделе подробно обсуждается математика, лежащая в основе открытых ключей. Для начала мы посмотрим, как получить открытый ключ из закрытого с использованием *умножения открытого ключа*. В последующих подразделах вы увидите, как осуществляется создание дочернего xpub с использованием *сложения открытого ключа* и как открытые ключи кодируются в системе Биткоин.

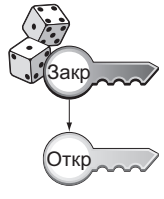
Умножение публичного ключа

ВНИМАНИЕ

Постараюсь объяснить эту тему максимально просто, но если вы сочтете это описание слишком сложным, то можете пропустить этот раздел и перейти к разделу «Повторение».

В главе 2 мы уже создавали открытый ключ из закрытого. Но я тогда не рассказал, *как* его создать. Попытаюсь восполнить этот пробел здесь.

Создание обычного открытого ключа



Открытый ключ в Биткоин — это целочисленное решение следующего уравнения:

$$y^2 = x^3 + 7 \bmod (2^{256} - 4294968273)$$

Это уравнение имеет множество решений, около 2^{256}, поэтому упростим задачу и используем решения уравнения $y^2 = x^3 + 7 \bmod 11$ (рис. 4.24).

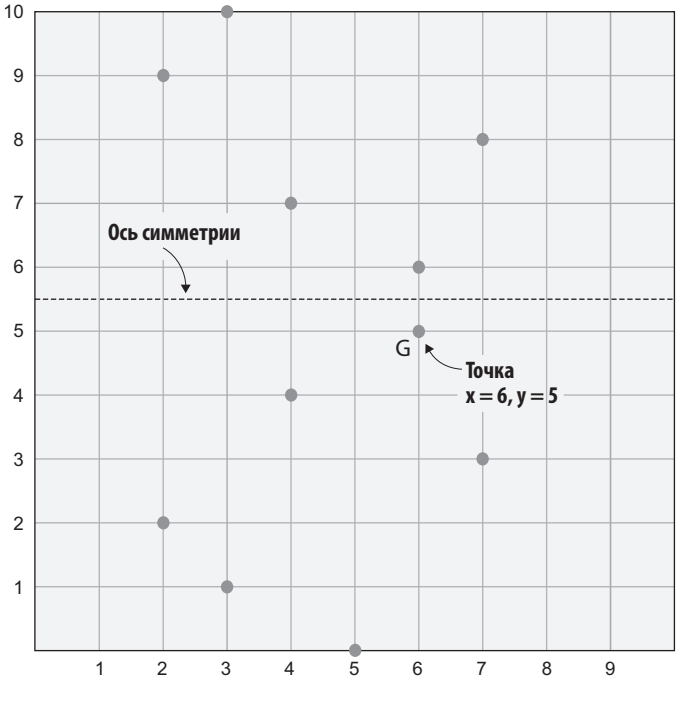


Рис. 4.24. Целочисленные решения эллиптической кривой $y^2 = x^3 + 7 \bmod 11$. Каждое решение — открытый ключ

Предыдущие формулы являются примерами класса уравнений, называемых *эллиптическими кривыми*, а их решение часто называют *точкой на кривой*. Теперь, имея закрытый ключ, мы можем вычислить открытый ключ, который является точкой на кривой. Для этого начнем с особой точки на кривой $G = (6,5)$. Вообще говоря, точка G была выбрана совершенно произвольно, но всем хорошо известно, что она служит началом для получения открытого ключа. *Открытый ключ — это закрытый ключ, умноженный на G.*

ЭТА КРИВАЯ ИСПОЛЬЗУЕТСЯ В БИТКОИН

Эта конкретная эллиптическая кривая называется *secp256k1* и используется в системе Биткоин. Есть множество других кривых, обладающих похожими свойствами.

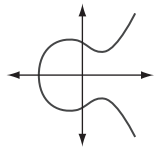
Если предположить, что закрытый ключ представлен числом 5, тогда открытым ключом будет число $5G$.

Чтобы найти это произведение, понадобятся две основные операции с открытым ключом: сложение и удвоение, где под удвоением подразумевается сложение точки с самой собой.

Чтобы сложить две точки (рис. 4.25), нарисуйте прямую линию, которая «перетекает» через края диаграммы и пересекает две ваши точки и одну третью точку. Эта третья точка является отрицательным результатом сложения. Чтобы получить истинный результат сложения, возьмите симметричную точку с тем же значением x.

КРИВАЯ? Я ВИЖУ ТОЛЬКО ТОЧКИ

Я говорю о *кривой*, потому что в непрерывном мире действительных чисел множество решений образуют кривую:



Результатом сложения $(6, 5) + (2, 2)$ является точка $(7, 8)$. Прямая, соединяющая две исходные точки, пересекает точку $(7, 3)$. Дополнением для точки $(7, 3)$ является точка $(7, 8)$ — результат сложения.

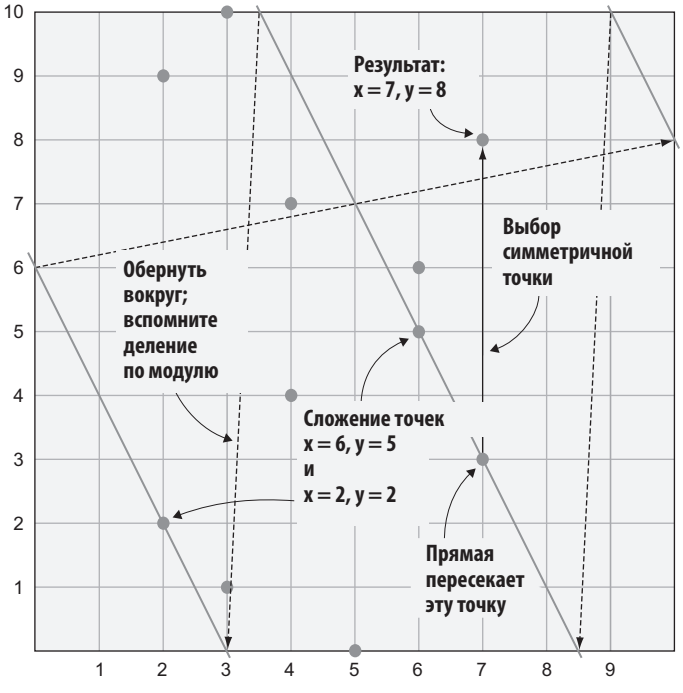


Рис. 4.25. Сложение точек. Чтобы сложить (x, y) = (6, 5) и (2, 2), нарисуйте прямую, соединяющую их и третью точку

Чтобы удвоить точку (рис. 4.26), нужно сложить ее с самой собой, но по одной точке нельзя узнать наклон прямой. В этом особом случае наклон прямой для единственной точки $P = (6,5)$ вычисляется как $3 \times x^2 \times (2y)^{-1} \bmod 11 = 2$. Процедура почти та же, как при сложении двух разных точек, только наклон прямой вычисляется иначе.

ВСЕГДА ЛИ СУЩЕСТВУЕТ ТРЕТЬЯ ТОЧКА?

Да, всегда есть третья точка, которую пересекает прямая. Это одно из важнейших свойств кривой.

С помощью этих двух основных операций, сложения и удвоения, можно выполнить умножение 5 и G. В двоичной форме 5 имеет вид

$$101_{\text{двоичное}} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0.$$

Тогда открытый ключ можно вычислить так:

$$5G = 1 \times 2^2 \times G + 0 \times 2^1 \times G + 1 \times 2^0 \times G.$$

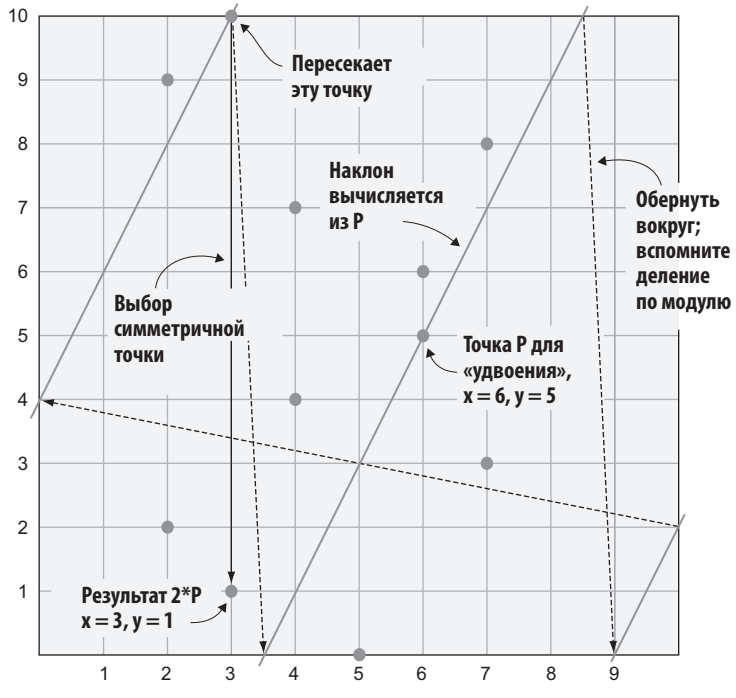


Рис. 4.26. Удвоение точки. Чтобы удвоить точку P, нарисуйте прямую, проходящую через точку P, с особым наклоном, который вычисляется из P. Прямая пересекает другую точку, (3,10). Дополнение для точки (3,1) является результатом удвоения

Начнем с точки G и вычислим точку открытого ключа, двигаясь по предыдущей формуле справа налево:

1. Вычислим $2^0 \times G = 1 \times G = G$. Все просто. Теперь запомним эту точку.
2. Вычислим $2^1 \times G = 2 \times G$. Это удвоение предыдущей точки G, которую мы вычислили и запомнили на шаге 1. Запомним и эту точку. Так произведение $2^1 \times G$ умножается на 0, мы ничего не будем делать с результатом — просто запомним его.
3. Вычислим $2^2 \times G = 2 \times 2 \times G$. Это удвоение предыдущей точки $2 \times G$. Так произ-

КАЛЬКУЛЯТОР ДЛЯ ВЫЧИСЛЕНИЯ ЭЛЛИПТИЧЕСКОЙ КРИВОЙ

На веб-ресурсе 11 (приложение B) имеется отличный калькулятор для вычисления эллиптических кривых, с которым вы можете поэкспериментировать, чтобы лучше понять механику вычислений.

ведение $2^2 \times G$ умножается на 1, прибавим этот результат к результату, полученному на шаге 1.

Как видите, умножение выполняется как последовательность операций удвоения и сложения.

Почему такой подход считается безопасным?

Выполнить умножение довольно просто; для 256-битного закрытого ключа нужно сделать около 256 шагов. Но обратные вычисления — это совсем другая история. Не существует известного способа получить закрытый ключ по точке «делением» (например, «разделить» точку (6,6) на G). Единственный известный способ — попробовать разные закрытые ключи и посмотреть, совпадает ли открытый ключ, полученный из них, с искомым. Именно это обстоятельство делает функцию создания открытого ключа однонаправленной.

Создание xpub

Теперь вы знаете, как путем умножения из закрытого ключа получается обычный открытый ключ. Но как сложить родительский открытый ключ с открытым ключом, полученным из левых 256 бит, чтобы получить дочерний открытый ключ? Взгляните на рис. 4.27.

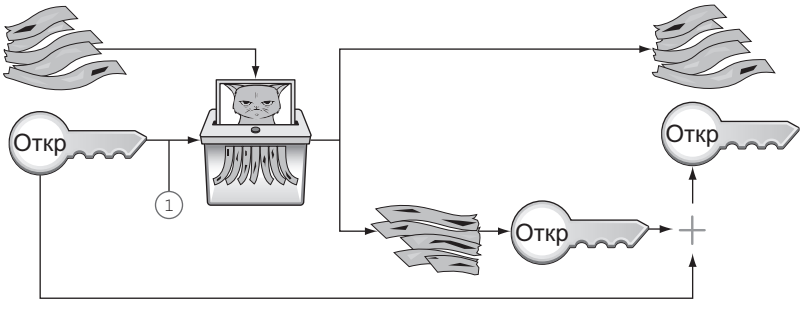


Рис. 4.27. Дочерний открытый ключ получается сложением родительского открытого ключа с открытым ключом, полученным из левых 256 бит

Убедиться, что это работает, можно, посмотрев, как создаются обычный и дочерний открытые ключи, на рис. 4.28.

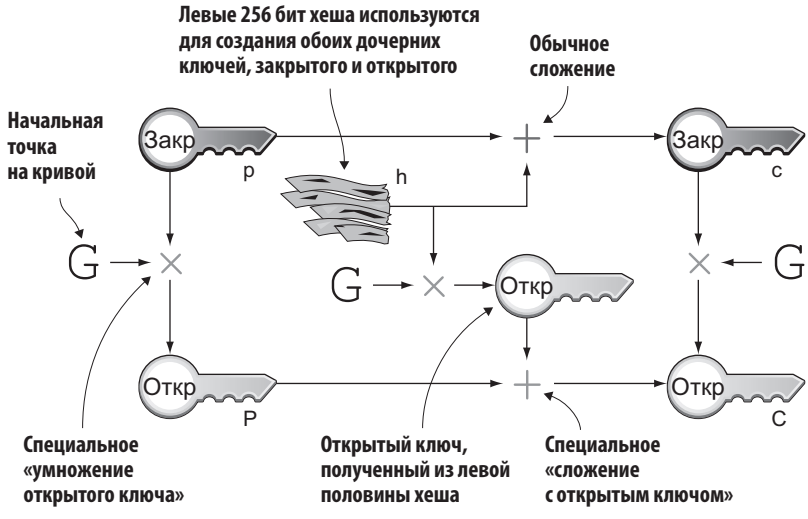


Рис. 4.28. Создание xpub и обычного открытого ключа. Обычный открытый ключ — это произведение начальной точки G на закрытый ключ. Дочерний открытый ключ — это сумма родительского открытого ключа и открытого ключа, полученного из левой половины хеша

Одна из замечательных особенностей эллиптических кривых состоит в том, что специальная операция «сложения» с открытым ключом немного напоминает обычное сложение. То же касается специального «умножения» открытого ключа. Остается только решить некоторые уравнения:

$$c = p + h,$$
$$C = Gh + Gp = G(h + p) = Gc.$$

Результат, $C = Gc$, точно описывает, как получить открытый ключ C из закрытого ключа c.

Кодирование открытого ключа

Надеюсь, вы помните, что открытый ключ Джона выглядит как большое число?

```
035541a13851a3742489fdddeef21be13c1abb85e053222c0dbf3703ba218dc1f3
```

Он явно не похож на пару координат, не так ли? Открытый ключ кодируется определенным образом. Вследствие симметрии каждому значению x соот-

ветствует ровно две точки: одна с четным значением y, а другая с нечетным (рис. 4.29).

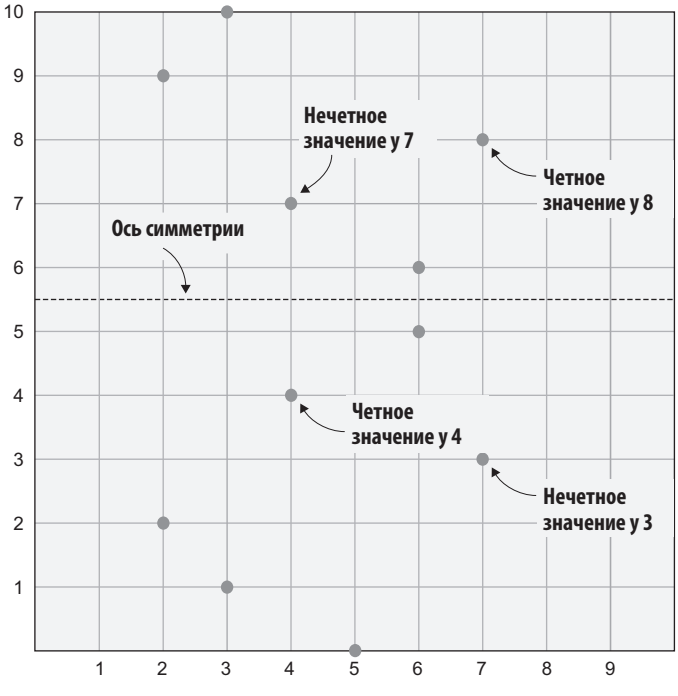


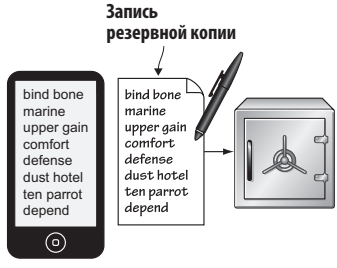
Рис. 4.29. Каждой точке на кривой соответствует симметричная точка с тем же значением x

Нет нужды запоминать значение y, достаточно знать, является ли оно четным или нечетным. Для этого к значению x добавляется префикс: `02` (для четного значения y) или `03` (для нечетного). В случае с Джоном y имеет нечетное значение, поэтому используется префикс `03`.

Вот почему открытые ключи имеют размер 33 байта, а не 32. Это 256-битное число — координата x — с префиксом, определяющим четность/нечетность.

Кривая на рис. 4.29 имеет одну точку $x = 5$, $y = 0$. Для нее не наблюдается симметричной точки. Это так называемый *двойной корень кривой* — две точки с одинаковым значением 0 координаты y. Они симметричны, потому что находятся на одинаковом расстоянии 5,5 от оси симметрии. В этом особом случае обе эти точки будут использовать префикс `02`, потому что 0 считается четным значением.

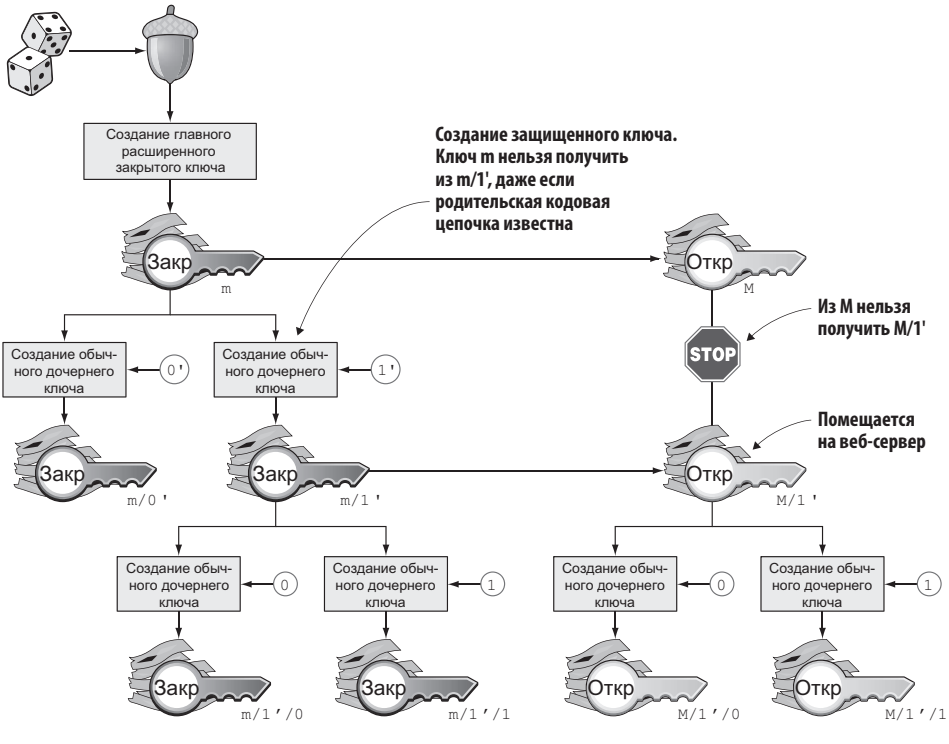
Повторение



Давайте повторим, что вы узнали в этой главе. HD-кошелек генерирует дерево ключей из случайного начального числа (семени). Он может использовать защищенные ключи, чтобы изолировать разные ветви дерева друг от друга.

Пользователи сохраняют резервные копии своих ключей, записывая начальное случайное число в виде мнемонического предложения из 12–24 английских слов на листе бумаги и запирая его в сейфе.

Кафе принимает жетоны на булочки в своем онлайн-магазине. Они помещают на веб-сервер только xpub для счета онлайн-продаж, `M/1'`, и получают возможность создавать произвольное количество адресов без использования закрытых ключей. Закрытые ключи хранятся в кошельке кафе и никогда не попадают на веб-сервер.



Изменения в системе

Наша таблица понятий (табл. 4.1) не изменилась в этой главе. Кошельки, описанные в этой главе, работают практически так же, как в системе Биткоин, но посылают электронное письмо Лизе, а не транзакции в глобальную сеть Биткоин. Мы обсудим этот вопрос в следующей главе.

Таблица 4.1. Ничего нового в таблице понятий

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
Электронная таблица	Блокчейн	Глава 6
Письмо Лизе	Транзакция	Глава 5
Запись в электронной таблице	Транзакция	Глава 5
Лиза	Майнер	Глава 7

А теперь закатим вечеринку! Из лаборатории выходит релиз 4.0 системы жетонов на булочки!

Таблица 4.2. Примечания к релизу, жетоны на булочки 4.0

Версия	Особенность	Как реализована
НОВАЯ 4.0	Простота платежей и создания новых адресов	Мобильное приложение «кошелек»
	Упрощенное резервное копирование	HD-кошельки генерируют ключи из начального случайного числа. Чтобы создать резервную копию, достаточно сохранить от 12 до 24 английских слов
	Создание адресов в небезопасном окружении	HD-кошельки способны генерировать деревья открытых ключей без использования закрытых ключей
3.0	Защищенность от дорогостоящих опечаток	Адреса в системе жетонов на булочки
	Улучшенная конфиденциальность	Вместо имен в электронной таблице сохраняются хеши открытых ключей (PKH)
2.0	Защищенная платежная система	Цифровые подписи решают проблему самозванцев
1.0	Простая платежная система	Основывается на доверии Лизе и личных знакомствах
	Конечный объем денег	Лиза получает в награду 7200 CT ежедневно; величина вознаграждения уменьшается вдвое каждые четыре года

Упражнения

Для разминки

4.1. Представьте, что вы пользуетесь приложением биткоин-кошелька и хотите получить 50 BTC от вашего друга на свой биткоин-адрес `155gWNamPrwKwu5D6JZdaLVKvxbpoKsp5S`. Сконструируйте платежный URI для передачи другу. Подсказка: в Биткоин такие URI начинаются с префикса `bitcoin:` вместо `ct:`. В остальном они ничем не отличаются.

4.2. Сколько раз нужно подбросить монету, чтобы получить случайный 10-символьный пароль из 64-символьного алфавита?

4.3. Назовите несколько проблем, характерных для резервных копий, защищенных паролем. Есть, по крайней мере, четыре такие проблемы.

4.4. Как создается начальное случайное число в HD-кошельке?

4.5. Из чего состоит xprv?

4.6. Из чего состоит xpub?

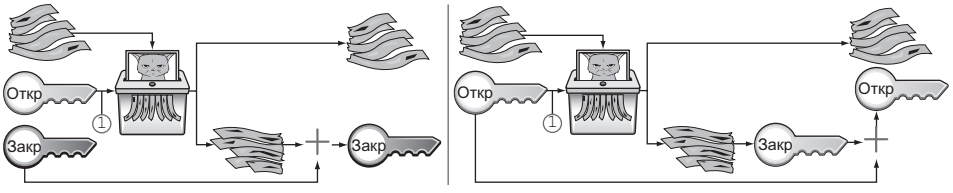
В упражнениях 4.7 и 4.8 предполагается, что вы прочитали раздел «Создание защищенных закрытых ключей». Если вы пропустили его, можете пропустить эти упражнения.

4.7. Вы решили создать защищенный xprv с индексом 7 из `m/2/1`. Какая информация понадобится вам для создания `m/2/1/7'`?

4.8. Можно ли создать xpub `M/2/1/7'` из `M/2/1`? Если нет, тогда опишите, как бы вы создали `M/2/1/7'`.

Придется пораскинуть мозгами

4.9. Предположим, вы не самый законопослушный гражданин и некоторым образом завладели главным xpub невежественной жертвы. Вы также украли закрытый ключ `m/4/1`, содержащий 1 BTC. Допустим, вы точно знаете, что закрытый ключ имеет именно этот конкретный путь. Опишите, как вы будете вычислять главный xprv. Используйте следующие подсказки:

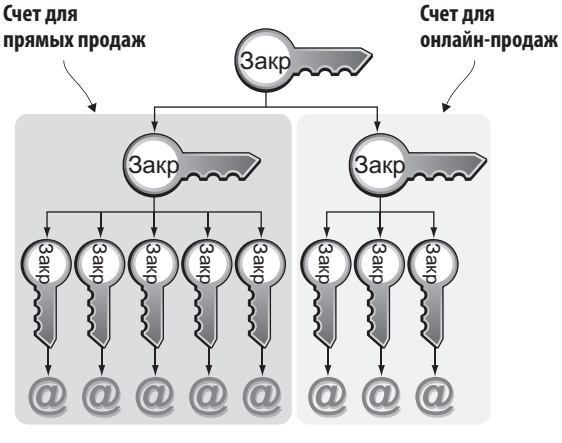


4.10. Теперь предположим, что ваша невежественная жертва имела 0 биткоинов на закрытом ключе `m/4/1`, но много денег на других адресах под тем же xprv. Сможете ли вы украсть эти деньги?

Если вы не читали раздел «Создание защищенных закрытых ключей», можете пропустить упражнение 4.11.

4.11. Предложите лучший подход, который ваша жертва могла бы использовать, чтобы помешать вам украсть все деньги.

4.12. Допустим, владелец кафе хочет дать своим сотрудникам доступ к счету прямых продаж, чтобы для каждой продажи они могли создать новый адрес. Но они не должны иметь доступа к закрытым ключам, потому что владелец не верит, что те смогут безопасно обращаться с ними. Подскажите, как этого добиться. Подсказка: кошелек может импортировать xpub.



4.13. Предположим, вы работаете в кафе и загрузили xpub в свой кошелек. Ваша коллега Анита загрузила тот же xpub в свой кошелек. Вы можете запрашивать у клиентов платежи, которые будут поступать на один и тот же счет. Как бы вы заметили, когда Анита получила деньги на ранее пустой ключ? Подсказка: ключи можно создать заранее.

Итоги

- Для отправки и получения денег — жетонов на булочки или биткоинов — обычно используется мобильное приложение, называемое кошельком.
- Кошелек создает и хранит ключи, сканирует или показывает реквизиты платежа, отправляет платежи, показывает ваш баланс и создает резервные копии ключей. Он избавляет от выполнения всех этих операций вручную.
- Выбрать правильный подход к резервному копированию трудно. Резервные копии, защищенные паролем, страдают от проблем, связанных с забытыми паролями, совершенствованием технологий, а также с отвратительными генераторами случайных чисел.
- HD-кошельки помогают создать резервную копию случайного начального числа и сохранить ее в безопасном месте. И сделать это только один раз.
- Начальное число можно преобразовать в мнемоническое предложение, чтобы упростить запись его на бумаге.
- HD-кошельки генерируют несколько закрытых ключей из начального числа и организуют их в древовидную структуру для увеличения конфиденциальности.
- Дерево открытых ключей — или любую из его ветвей — можно сгенерировать из xpub. Эту особенность удобно использовать в небезопасных окружениях, таких как веб-серверы.
- Создание защищенных закрытых ключей обеспечивает разделение «счетов». Этот прием ограничивает возможность взлома единственным счетом.

5 Транзакции

Эта глава охватывает следующие темы:

- ✓ транзакции с биткоинами или жетонами на булочки;
- ✓ создание, подтверждение и проверка транзакций;
- ✓ программирование денег.

Оплату булочек жетонами, которую вы с коллегами до сих пор осуществляли, имеет ряд серьезных проблем. Худшая — Лиза может не устоять перед искушением и совершить кражу, что очень волнует новичков. Они не решаются пользоваться системой, предполагая, что Лиза способна на воровство.

Основное внимание в этой главе мы уделим *транзакциям* (рис. 5.1): фрагментам данных, которые формализуют порядок выполнения платежей через Лизу. Транзакции заменят электронные письма Лизе. Они будут храниться в электронной таблице в своем естественном виде вместо данных в столбцах «Кому», «От кого» и «СТ». Это лишит Лизу возможности красть деньги других людей, потому что теперь любой сможет проверить все платежи в электронной таблице.

В этой главе мы углубимся в исследование транзакций и посмотрим, насколько они *программируемые*, то есть гибкие, и что с ними можно делать. Например, транзакции с мультиподписью могут потребовать две подписи из трех возможных, чтобы потратить деньги, общие для трех человек.

К концу этой главы система сильно изменится в том, как кошельки создают платежи, как Лиза проверяет платежи и как они хранятся. Но самое главное, каждый сможет проверить платежи в электронной таблице.

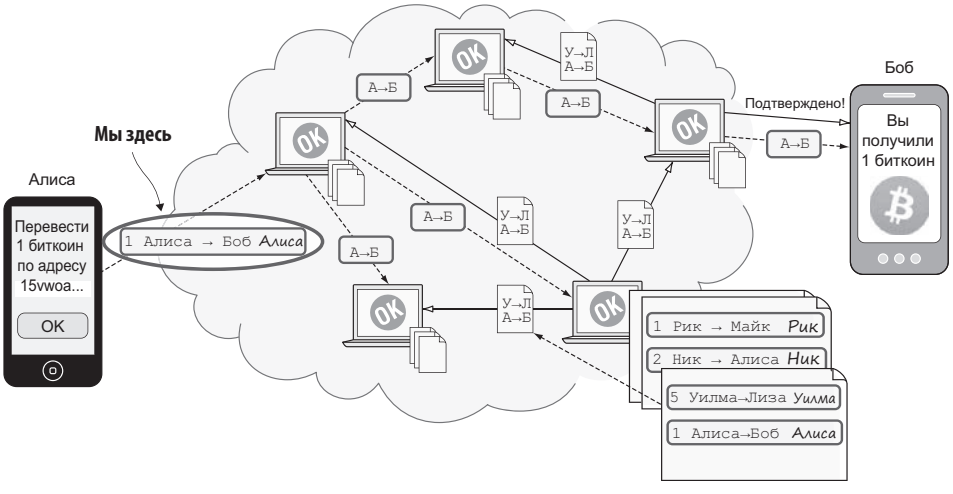


Рис. 5.1. Транзакции в Биткоине

Проблемы в старой системе

Лиза выполняет важную работу. Она следит за тем, чтобы никто не мошенничал, проверяя цифровую подпись и наличие средств на хеше открытого ключа (PKH) перед подтверждением платежа. Она подтверждает платежи, добавляя их в электронную таблицу.

Но этот подход страдает несколькими проблемами:

- Лизе надоело подсчитывать баланс перед подтверждением каждого платежа. Таблица растет, и каждая проверка занимает все больше и больше времени.
- Если у вас есть два адреса с 5 CT на каждом, вам придется выполнить два отдельных платежа, чтобы заплатить 10 CT за булочки. Это ложится ненужным бременем на отправителя и Лизу, а также увеличивает размер электронной таблицы, добавляя избыточные записи.
- Компания растет, и в ней появляются новые работники, плохо знающие Лизу, из-за чего доверие к ней начинает снижаться. Некоторые боятся, что Лиза украдет их жетоны в электронной таблице. Только Лиза может проверять подписи, потому что только она видит отправленные ей электронные письма. Поэтому она, например, *могла бы* увеличить сумму в столбце «CT» платежа, отправленного ей, или добавить запись с ложным платежом, скажем, от Джона Лизе (рис. 5.2). При этом никто не сможет доказать, что Лиза совершила мошенничество. Неважно, что

она заслуживает доверия больше, чем кто-либо. Люди, не знающие ее, будут думать, что Лиза такая же жадная, как и все остальные.

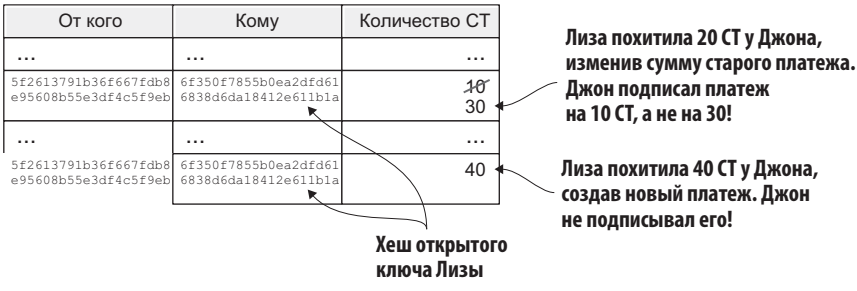


Рис. 5.2. Виды мошенничества, которые могла бы совершить Лиза. Она этого не делает, но имеет такую возможность

Обратите внимание, что Лиза не может создавать новые деньги, кроме 7200 СТ в день, как было согласовано изначально. Также, если она попытается украсть больше, чем доступно на РКН, кто-то, проверяющий электронную таблицу, может заметить, что общая сумма денег слишком велика, и Лизу уличат в воровстве.

МИНИМИЗАЦИЯ НЕОБХОДИМОСТИ ДОВЕРИЯ

Отсутствие необходимости полагаться на доверие — вот что такое Биткоин. Транзакции еще на один шаг приближают нас к надежной системе, в которой каждый может проверить все.

Лизу огорчает недоверие других. Но она понимает, что почти ничего не может сделать, чтобы изменить степень доверия к себе. Интересная альтернатива — *минимизировать необходимость доверия*. Она подумала, что для этого лучше всего сделать процесс сверхпрозрачным, чтобы каждый мог проверять платежи. Одновременно с этим упростится проверка наличия средств у людей и появится возможность тратить деньги сразу с нескольких адресов. Она изобрела *транзакцию с жетонами* и смогла решить все три проблемы, описанные выше.

Платежи с использованием транзакций

Транзакции меняют модель отправки платежа Лизе и его хранения в электронной таблице. Они не меняют поведения кошельков с точки зрения пользователя — приложение кошелька будет выглядеть точно так же.

Предположим, Джон хочет купить булочку. Но теперь ему не нужно писать письмо Лизе, как он делал это раньше. Теперь кошелек Джона использует транзакции, поэтому он просто создаст транзакцию, как показано на рис. 5.3. Цель транзакции — перевести 10 CT на адрес кафе.

Рис. 5.3. Процедура оплаты для пользователей осталась прежней, но изменилась для Лизы и электронной таблицы

Джон сканирует платежный URI кафе, затем его кошелек создает транзакцию и просит подтвердить ее. Джон нажимает кнопку OK, и кошелек подписывает транзакцию. Затем кошелек отправляет подписанную транзакцию как вложение в пустом электронном письме Лизе.

Транзакция содержит информацию о том, куда отправить деньги, а также о том, *сколько денег* потратить, ссылаясь на конкретные «монеты», называемые *неизрасходованными выходами транзакций* (Unspent Transaction Outputs, UTXO), полученные Джоном в предыдущих входящих транзакциях.

Лиза убеждается, что монеты в исходящей транзакции действительно существуют и еще не потрачены. Она также проверяет действительность подписей (в транзакции их может быть несколько). Если все проверки успешно пройдены, Лиза подтвердит транзакцию, добавив ее в конец таблицы в том виде, в каком она была получена.

После того как транзакция попадет в электронную таблицу, любой желающий сможет выполнить ту же проверку, что и Лиза, и убедиться, что Лиза ни у кого ничего не крадет и вообще никак не касается денег других людей.

В следующих трех подразделах мы подробнее рассмотрим три перечисленных этапа: создание, подтверждение и проверка.

Создание транзакции

Давайте рассмотрим поближе, как создается транзакция Джона.

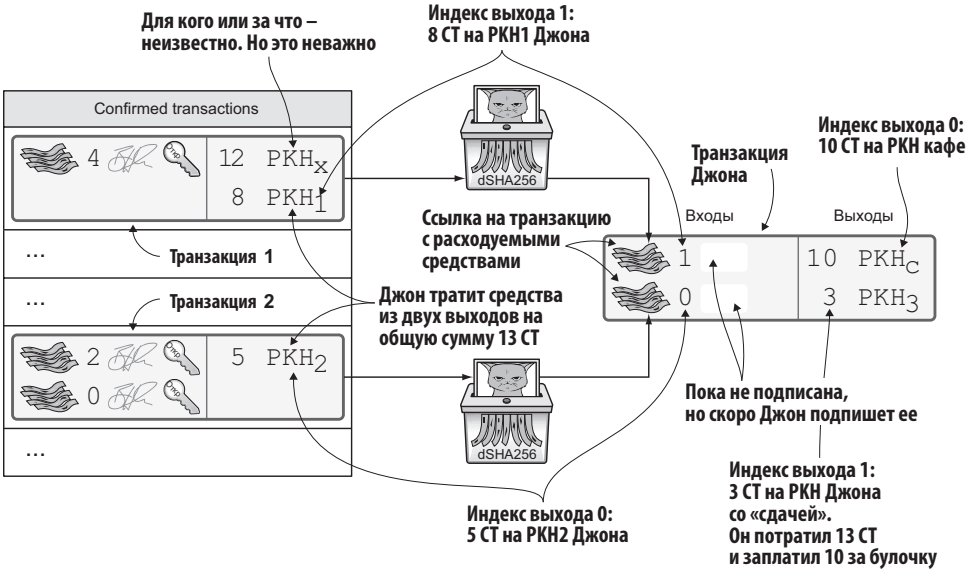


Рис. 5.4. Кошелек Джона подготавливает платеж на 10 CT за булочку. Для покрытия расходов он использует два ключа, на которых имеются средства. Он перекладывает свои 3 CT на свой новый адрес. Транзакция пока не подписана

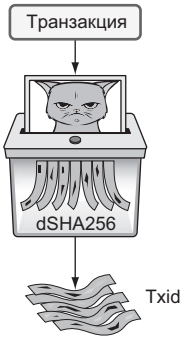
Кошелек Джона создал новую транзакцию (рис. 5.4). Она имеет два входа и два выхода. Входы указывают на неизрасходованные выходы транзакций, которыми осуществляется оплата, а выходы — куда идут деньги.

ТРАНЗАКЦИЯ ДЖОНА

- ☐ Создание (Джон)
- ☐ Подтверждение (Лиза)
- ☐ Проверка (любой желающий)

Входы

Входы определяют транзакции с неизрасходованными выходами, которые должны быть потрачены. У Джона есть два неизрасходованных выхода (UTXO), один с 8 CT и другой с 5 CT. Неизрасходованные выходы принадлежат двум предыдущим транзакциям, 1 и 2, посредством которых Джон получил деньги. Теперь Джон хочет потратить эти UTXO.



Вход транзакции ссылается на предыдущую транзакцию, указывая ее *идентификатор* (txid). Идентификатор транзакции txid — это двойной хеш SHA256. Он называется идентификатором, потому что часто используется для ссылки на транзакцию, как во входах на рис. 5.4.

ПРИМЕЧАНИЕ

Обоснование использования двойного SHA256 здесь не совсем понятно, но, по всей видимости, оно предотвращает атаку удлинением сообщения. Создатель Биткоин, вероятно, использовал двойной SHA256 как меру безопасности, чтобы не думать об атаках такого рода. Подробнее см. веб-ресурс 12 (приложение B).

Первый вход в транзакции Джона, с индексом 0, содержит:

- идентификатор txid транзакции 1;
- индекс 1 расходуемого выхода из транзакции 1;
- пустое место для подписи.

Второй вход в его транзакции, с индексом 1, содержит:

- идентификатор txid транзакции 2;
- индекс 0 расходуемого выхода из транзакции 2;
- пустое место для подписи.

Джон добавит подписи позже, когда закончится создание транзакции.

Выходы

Выход транзакции содержит сумму и PKH. Транзакция Джона имеет два выхода. Выход с индексом 0 описывает плату 10 CT на PKH_C кафе за булочку.

Выход с индексом 1 описывает возврат сдачи 3 CT на один из собственных ключей Джона, PKH_3. Мы называем эту операцию *сдачей*, потому что она напоминает традиционную сдачу, когда вы платите 75 долларов 100-долларовой купюрой и получаете 25 долларов назад: Джон платит 13 CT и возвращает 3 CT на свой адрес для сдачи PKH_3.

Сдача необходима, потому что нельзя потратить только часть неизрасходованного выхода транзакции. Вы должны потратить его полностью или не потратить вообще.

Выходы и входы намного сложнее, чем может показаться: они содержат не только PKH в выходе и подпись во входе. В действительности выход содержит компьютерную программу, которая проверит подпись во входе. Мы поговорим об этом позже.

Чтобы транзакция была действительной, сумма во входах должна быть больше или равна сумме в выходах. Разница, если таковая имеется, называется *комиссией за транзакцию*, о которой мы поговорим в главе 7. Пока Джон не платит комиссию за транзакцию, поэтому его сумма на выходе точно соответствует сумме на входе.

КОМИССИЯ ЗА ТРАНЗАКЦИЮ

Обычно вы должны заплатить сети Биткоин комиссию за обработку транзакции.

Теперь транзакция создана, но еще не подписана. Любой может создать такую транзакцию, потому что она основана на общедоступной информации. Входы просто ссылаются на транзакции в электронной таблице и индексы внутри них. Но только Джон сможет подписать эту транзакцию, потому что только у него есть закрытые ключи, соответствующие PKH_1 и PKH_2.

Подписывание транзакции

Чтобы подтвердить транзакцию, Джон нажимает кнопку OK в своем кошельке. После этого кошелек должен создать две подписи, по одной для PKH_1 и PKH_2, а для этого у Джона должны иметься оба закрытых ключа — для PKH_1 и для PKH_2 (рис. 5.5).

Каждый вход должен подписываться отдельно. Вход с индексом 0 должен быть подписан закрытым ключом, соответствующим хешу PKH_1, потому что этот вход описывает оплату деньгами с адреса PKH_1. Аналогично вход с индексом 1 должен быть подписан закрытым ключом, соответствующим хешу PKH_2, потому что описывает оплату деньгами с адреса PKH_2.

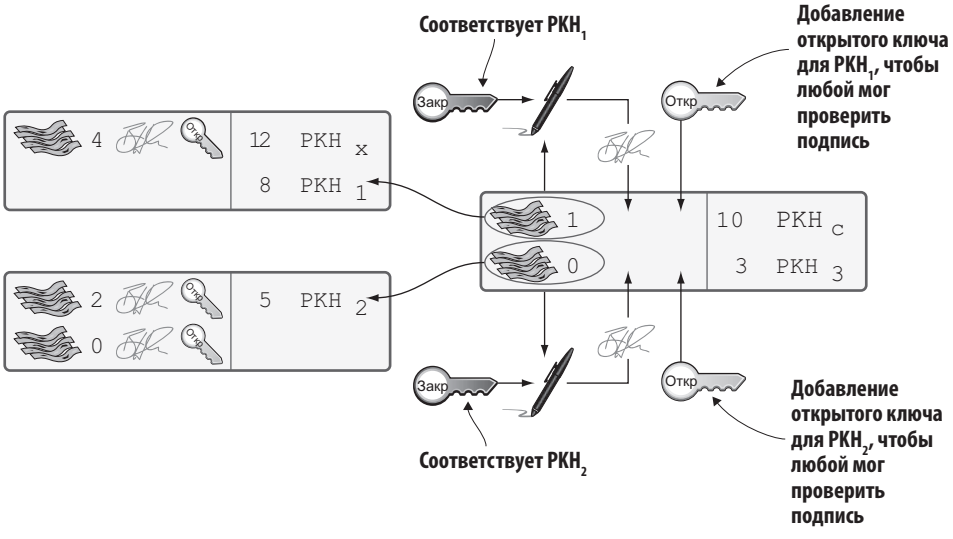


Рис. 5.5. Кошелек Джона подписывает транзакцию. Каждый вход получает отдельную подпись. Во входных данных также должен присутствовать открытый ключ, чтобы любой мог проверить подпись

Каждая подпись подтверждает всю транзакцию, то есть алгоритм создания подписи будет хешировать всю транзакцию, кроме самих подписей. Если что-то изменится в содержимом транзакции, любая подпись для этой транзакции станет недействительной.

Для упрощения проверки подписывается чистая версия транзакции, то есть версия без подписей во входах. Нельзя добавить подпись во вход 0, *а затем* во вход 1. Проверка будет затруднена, если проверяющий не знает порядок, в каком добавлялись подписи. Когда *все* подписи создаются на основе чистой транзакции *и только потом* добавляются в нее, порядок создания подписей не имеет значения.

После создания подписей кошелек добавляет их в транзакцию. Но пока остается неясным еще один аспект: как кто-то, проверяющий транзакцию, например кафе, узнает открытый ключ для проверки подписи? Проверяющий в кафе может видеть только хеш РКН в выходе и подпись во входе. Он не может получить открытый ключ для РКН, потому что, как вы наверняка помните, криптографические хеши являются однонаправленными функциями. Кошелек Джона должен явно добавить соответствующий открытый ключ во вход. Подпись во входе 0, которая тратит деньги из $РКН_1$, должна проверяться с помощью открытого ключа, из которого был сгенерирован

хеш $РКН_1$. Аналогично во вход 1 добавляется открытый ключ, соответствующий $РКН_2$.

Лиза подтверждает транзакцию

Транзакция готова к отправке Лизе. Кошелек Джона отправляет ее во вложении в электронном письме. Лиза извлекает транзакцию и убеждается, что:

ТРАНЗАКЦИЯ ДЖОНА

- ☑ Создание (Джон)
- ☐ Подтверждение (Лиза)
- ☐ Проверка (любой желающий)

- ✶ Транзакция расходует средства транзакций, которые действительно существуют в электронной таблице и еще не были потрачены какой-либо другой транзакцией в электронной таблице.
- ✶ Сумма выходов транзакции не превышает сумму входов. Если это правило не соблюсти, транзакция создаст новые деньги из воздуха.
- ✶ Подписи Джона верны.

Лизе больше не нужно вычислять баланс РКН, но она должна убедиться, что деньги, которые предполагается потратить, действительно существуют и еще не были потрачены.

Как она проверит, что средства, указанные в выходе транзакции, не израсходованы? Может быть, она должна попытаться отыскать в электронной таблице транзакции, которые тратят эти средства? Именно так она и поступает. Этот поиск кажется таким же обременительным, как поиск для расчета баланса. Не волнуйтесь: у Лизы есть свой план.

Множество неизрасходованных входящих выходов

Чтобы упростить проверку неизрасходованных средств, она создает новую закрытую базу данных, которую она называет *множеством UTXO* (рис. 5.6). Это набор всех UTXO.

МНОЖЕСТВО UTXO

Все узлы в сети Биткоин поддерживают закрытое множество UTXO, чтобы ускорить проверку транзакций.

Запись в множестве UTXO включает идентификатор транзакции txid, индекс и фактический выход транзакции. Лиза хранит свое множество UTXO для проверки транзакций. Прежде чем добавить транзакцию Джона

в электронную таблицу, Лиза проверяет присутствие в множестве UTXO выходов, которые тратит транзакция. Если их там нет, значит, Джон пытается потратить деньги, которые никогда не существовали в электронной таблице или уже потрачены (обычно это называется *попыткой двойного расходования*).

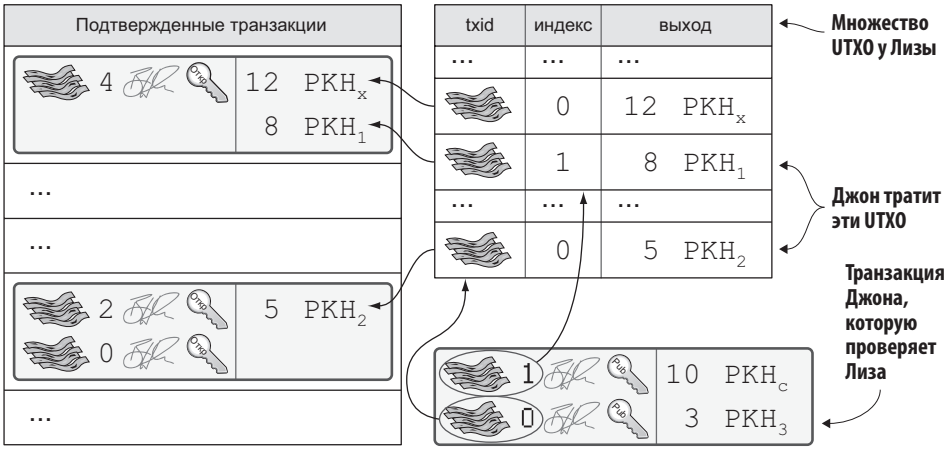


Рис. 5.6. Используя свое множество UTXO, Лиза убеждается, что Джон не попытался дважды потратить одни и те же деньги

Лиза использует свое множество UTXO для проверки каждого входа в транзакции Джона, выполнив поиск идентификаторов транзакций и индексов выходов. Если все расходуемые выходы присутствуют в множестве UTXO, значит, эта транзакция не является попыткой двойного расходования или расходования несуществующих средств. В данном случае Лиза обнаружит оба выхода в своем множестве UTXO и затем перейдет к проверке подписей в обоих входах транзакции Джона.

ДВОЙНОЕ РАСХОДОВАНИЕ

Под двойным расходованием понимается попытка дважды потратить один и тот же выход. Лиза может предотвратить двойное расходование, выполнив проверку в своем множестве UTXO.

Она извлекает PKH выхода из первого входа и сравнивает с хешем открытого ключа, указанного в этом же входе (рис. 5.7). Затем проверяет подпись во входе, используя открытый ключ, подпись и транзакцию, а затем тем же способом проверяет подпись второго входа. Обе проверки завершаются успехом.

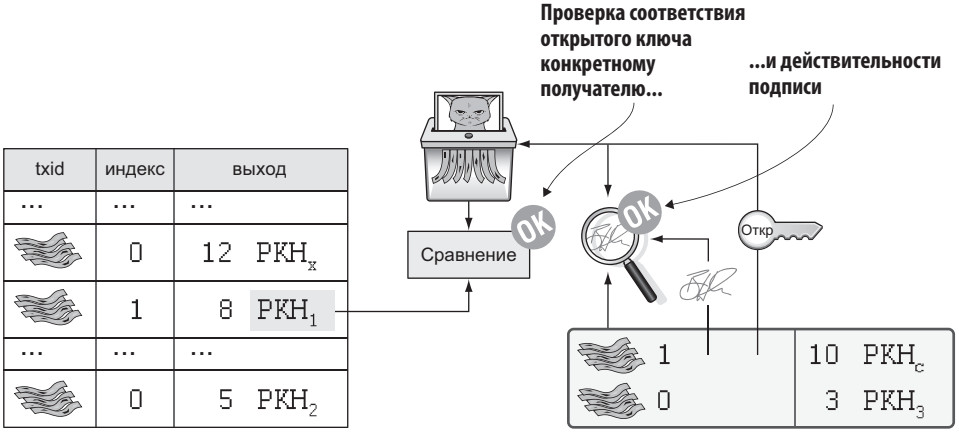


Рис. 5.7. Лиза проверяет первую подпись транзакции Джона

Затем Лиза добавляет подтвержденную транзакцию в электронную таблицу. Она должна удалить потраченные выходы из набора UTXO и добавить туда выходы из транзакции Джона (рис. 5.8). Именно так она поддерживает соответствие множества UTXO содержимому электронной таблицы транзакций.

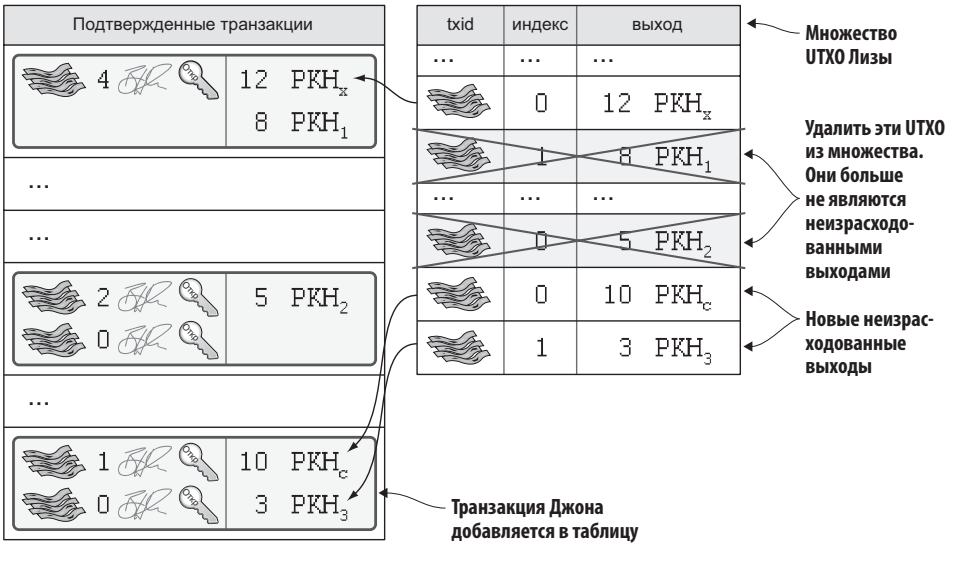


Рис. 5.8. Лиза добавляет транзакцию в электронную таблицу и удаляет израсходованные выходы прежних транзакций из множества UTXO

Лиза поддерживает множество UTXO в актуальном состоянии, обновляя его, как показано на рис. 5.8, согласно входящей транзакции. Потеряв набор UTXO, Лиза сможет воссоздать его из электронной таблицы, начав с пустого набора UTXO и повторно применив к нему все транзакции в электронной таблице, одну за другой.

Создать множество UTXO может не только Лиза. Любой, имеющий доступ к электронной таблице, сможет сделать то же самое. Это пригодится нам в следующих главах, когда мы заменим Лизу несколькими людьми, выполняющими ее работу. Для людей, которые просто хотят проверить электронную таблицу, тоже важно убедиться, что информация в ней верна.

ПОВТОРНОЕ СОЗДАНИЕ МНОЖЕСТВА UTXO

Множество UTXO создается только из транзакций в электронной таблице. Его можно воссоздать в любой момент, при наличии доступа к электронной таблице.

Любой может проверить транзакцию

Теперь, когда транзакция Джона сохраняется в электронной таблице в исходном виде, любой имеющий доступ для чтения сможет ее проверить. Любой может создать свое *закрытое* множество UTXO, в точности соответствующее множеству UTXO Лизы, просто выполнив все транзакции.

ТРАНЗАКЦИЯ ДЖОНА

- ☑ Создание (Джон)
- ☑ Подтверждение (Лиза)
- ☐ Проверка (любой желающий)

То есть любой сможет выполнить те же проверки, что и Лиза. Любой сможет убедиться, что Лиза добросовестно делает свою работу. Эта возможность важна для системы, потому что позволяет убедиться, что таблица заполняется в полном соответствии с установленными правилами.

В Биткоин такие узлы, осуществляющие проверку, называют *полными узлами*. Лиза тоже является полным узлом (верификатором), но она делает чуть больше, чем полный узел, — она обновляет электронную таблицу. В Биткоин полный узел также называют *проверяющим узлом* или, неформально, просто *узлом*.

ТРАНЗАКЦИЯ ДЖОНА

- ☑ Создание (Джон)
- ☑ Подтверждение (Лиза)
- ☑ Проверка (любой желающий)

Лиза больше не может украсть чужие деньги, потому что кража сделает таблицу недействительной. Например, допустим, она попыталась изменить получателя в выходе транзакции Джона с $РКН_С$ на $РКН_L$. Фактически это попытка украсть 10 CT у кафе (рис. 5.9).

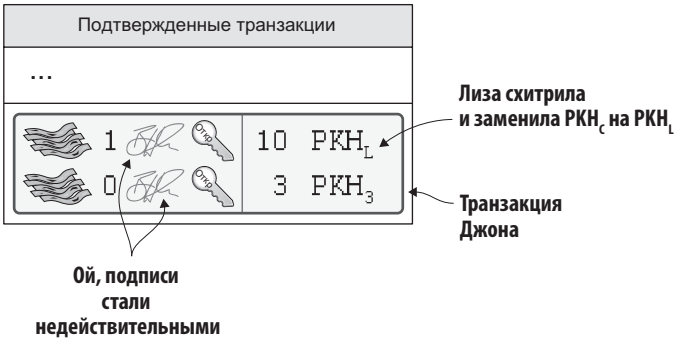


Рис. 5.9. Лиза больше не имеет возможности украсть чужие деньги. Если она это сделает, подписи станут недействительными и ее аморальный поступок раскроется

Поскольку Лиза изменила содержимое транзакции Джона, подписи в этой транзакции станут недействительными. Любой, имеющий доступ к электронной таблице, заметит это, потому что все очень прозрачно.

Следствия безопасности общедоступных подписей

Возможность проверить все транзакции любым желающим — одна из замечательных особенностей общедоступных подписей. Но они имеют небольшой недостаток.

Вспомните, как в главе 3 мы вводили в обращение хеш открытого ключа РКН. Когда мы использовали РКН, открытый ключ не указывался в электронной таблице. Это обеспечивало двухуровневую защиту денег: с помощью функции получения открытого ключа и криптографической хеш-функции (SHA256 + RIPEMD160). Даже если открытый ключ будет кем-то вскрыт, закрытый ключ останется защищенным функцией создания открытого ключа. Это напоминает избыточную перестраховку.

Но при использовании транзакций открытый ключ присутствует в открытом виде во входах транзакций, расходующих выходы других транзакций. Взгляните еще раз на транзакцию Джона на рис. 5.10.

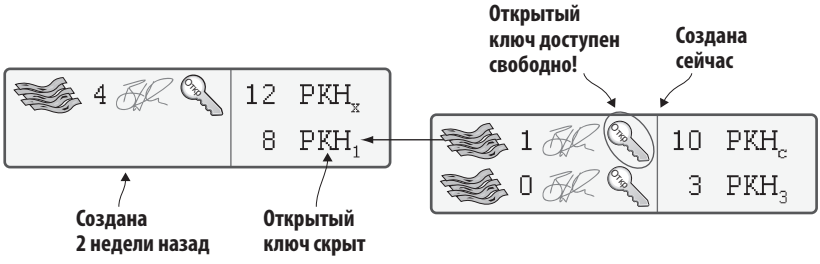


Рис. 5.10. Открытый ключ свободно доступен во входе транзакции. В главе 3 мы прикладывали дополнительные усилия, чтобы не допустить этого

Вход содержит открытый ключ. Но он становится общедоступным только после расходования выходов. В связи с этим можно дать важный совет: не используйте адреса повторно! Если у Джона есть другие неизрасходованные выходы на PKH_1, теперь они окажутся менее защищенными, потому что больше не защищены криптографической хеш-функцией — только функцией создания открытого ключа.

НЕ ИСПОЛЬЗУЙТЕ АДРЕСА ПОВТОРНО

Биткоин-адреса не следует использовать повторно. Повторное использование ухудшает защищенность и конфиденциальность.

Повторное использование адресов не только ослабляет безопасность закрытых ключей, но также ухудшает конфиденциальность, как обсуждалось в главе 3. Снова предположим, что у Джона есть другие неизрасходованные выходы с адресом PKH_1. Если Acme Insurances вынудит кафе раскрыть информацию о том, что данная транзакция описывает покупку булочки Джоном, в Acme также смогут узнать, что все выходы с адресом PKH_1 принадлежат Джону. Это касается и выходов со сдачей.

К счастью, кошельки автоматизируют создание ключей, поэтому обычно не приходится беспокоиться о многократном использовании одних и тех же ключей. В настоящее время большинство биткоин-кошельков, имеющихся на рынке, используют уникальные адреса для всех входящих платежей.

Системы на основе счетов и на основе ценностей

Давайте вспомним изменения, которые мы сделали. Мы перешли от системы *на основе счетов* к системе *на основе ценностей*.

Система на основе счетов следит за суммами, лежащими на каждом счете. Систему этого типа мы имели в начале главы. Лиза должна была вычислять баланс PKH, прежде чем подтвердить платеж.

Система на основе ценностей, напротив, следит за «монетами». В этой главе Лиза должна проверить наличие определенных монет (UTXO), прежде чем подтвердить платеж. Ей не нужно проверять баланс любого PKH. Система Биткоин тоже основана на ценностях.

Язык сценариев

Я был несколько неточен, описывая содержимое транзакции. Выход транзакции содержит не PKH, а часть небольшой компьютерной программы, *включающей* PKH. Эта часть программы называется *сценарием открытого ключа* (script pubkey). Вход, расходующий этот выход, содержит другую часть программы. Эта другая часть, подпись и открытый ключ в транзакции Джона, называется *сценарием подписи* (signature script), как показано на рис. 5.11.

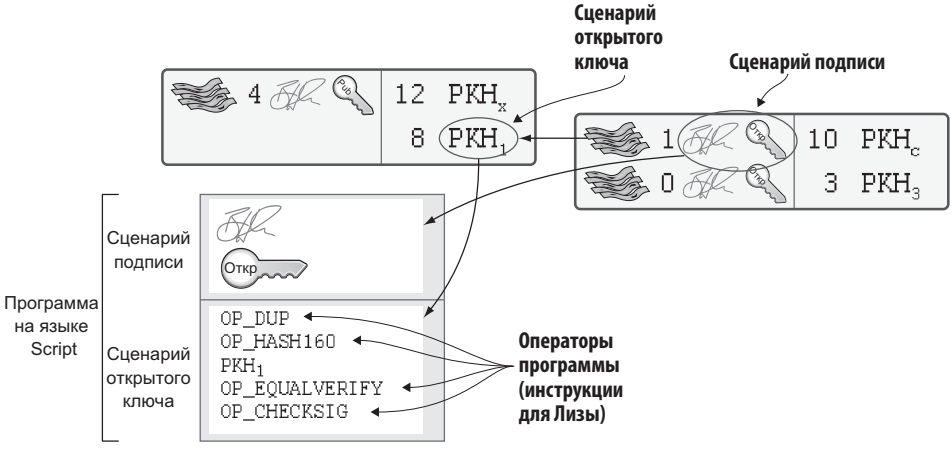


Рис. 5.11. Сценарий подписи — это первая часть программы. Сценарий открытого ключа в выходе транзакции, описывающей платеж, — вторая ее часть. Если полная программа завершится успехом, транзакции будет позволено потратить выход

Эта крохотная программа, написанная на языке сценариев (Script), содержит инструкции для Лизы, описывающие, как проверить подлинность

транзакции, расходующей средства. Если Лиза безошибочно выполнит все инструкции в программе и получит в результате OK, то транзакция будет считаться подлинной.

Возможность добавления компьютерных программ внутрь транзакций может пригодиться в самых разных случаях использования. В этой книге мы рассмотрим несколько вариантов применения специализированных программ.

Допустим, Лиза решила проверить вход 0 транзакции Джона. Она должна выполнить инструкции в программе сверху вниз. Для хранения промежуточных результатов используется *стек*. Стек напоминает стопку листов бумаги. Вы можете положить лист на вершину стопки и снять лист с вершины.

Итак, взгляните на рис. 5.12. Первый (верхний) элемент в программе — это подпись, которая играет роль данных. Когда в программе встречаются данные, они просто переносятся в стек. Лиза переносит подпись на вершину изначально пустого стека. Затем она встречает открытый ключ, который тоже играет роль обычных данных. Она кладет этот ключ на вершину стека. Теперь стек содержит подпись и открытый ключ, причем открытый ключ лежит сверху.

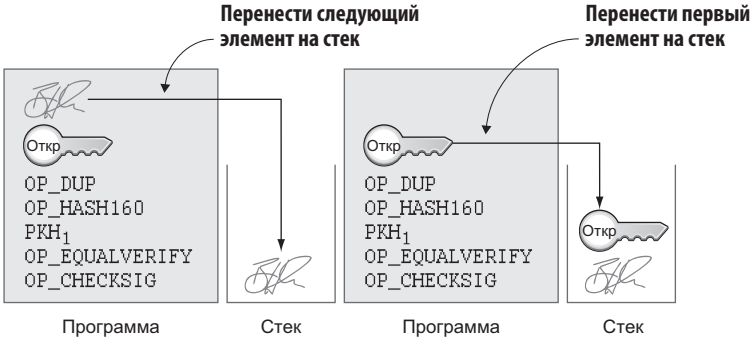


Рис. 5.12. Подпись и открытый ключ переносятся на стек

Следующий элемент в программе — OP_DUP (рис. 5.13). Это уже не данные, а оператор. Оператор производит вычисления с элементами в стеке и, в некоторых случаях, с проверяемой транзакцией. Этот конкретный оператор довольно прост: он означает «Скопировать верхний элемент в стеке (не удаляя его) и поместить копию сверху». Лиза выполняет инструкцию и по-

мещает копию открытого ключа на вершину стека. Теперь в стеке у нас имеются два открытых ключа и подпись.

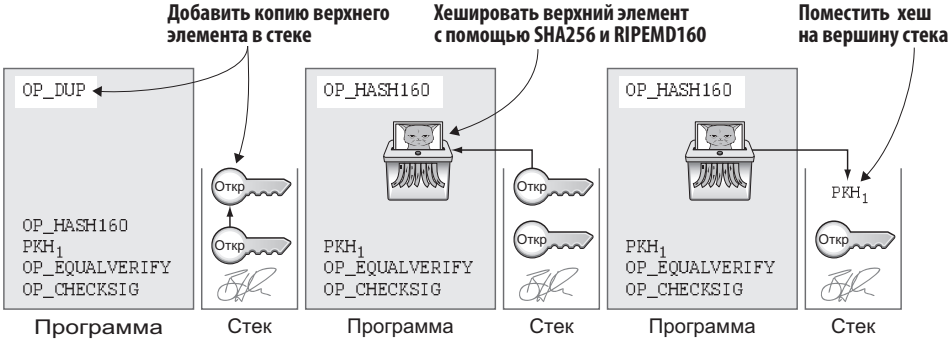


Рис. 5.13. Копирование открытого ключа в стеке и добавление PKH

Следующий элемент программы, `OP_HASH160`, — еще один оператор (он также показан на рис. 5.13). Он означает «Снять верхний элемент со стека, хешировать его с использованием пары функций SHA256+RIPEMD160 и положить результат на стек».

Отлично. Лиза извлекает открытый ключ из стека, хеширует его и помещает получившийся PKH на вершину стека. В результате получился PKH_1 Джона, потому что Лиза хешировала открытый ключ Джона.

Следующий элемент программы — просто данные (рис. 5.14): это PKH_1, представляющий получателя 8 CT. Лиза переносит PKH_1 на стек.

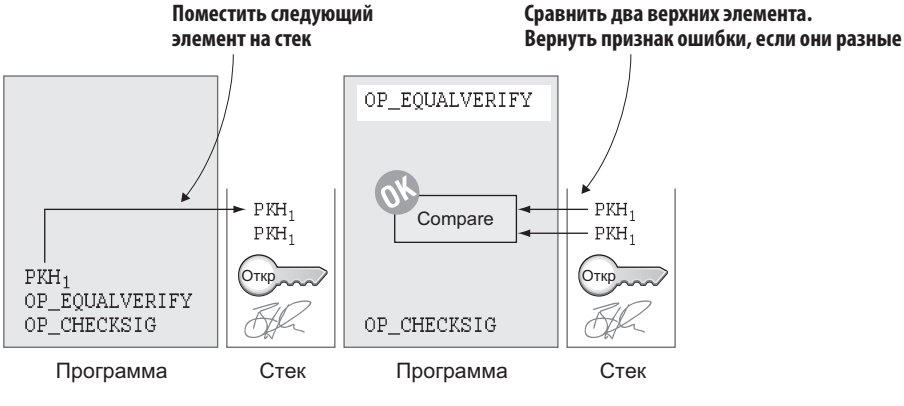


Рис. 5.14. Добавление PKH_1 на стек и сравнение двух PKH

Далее следует еще один оператор, `OP_EQUALVERIFY`. Он означает «Снять два верхних элемента со стека и сравнить их. Если они равны, перейти к следующей инструкции; иначе завершить программу с признаком ошибки». Лиза снимает два элемента PKH с вершины стека и сравнивает их. Они *равны*, а значит, открытый ключ, который передал Джон в сценарии подписи в своей транзакции, соответствует PKH, заданному в качестве получателя выхода.

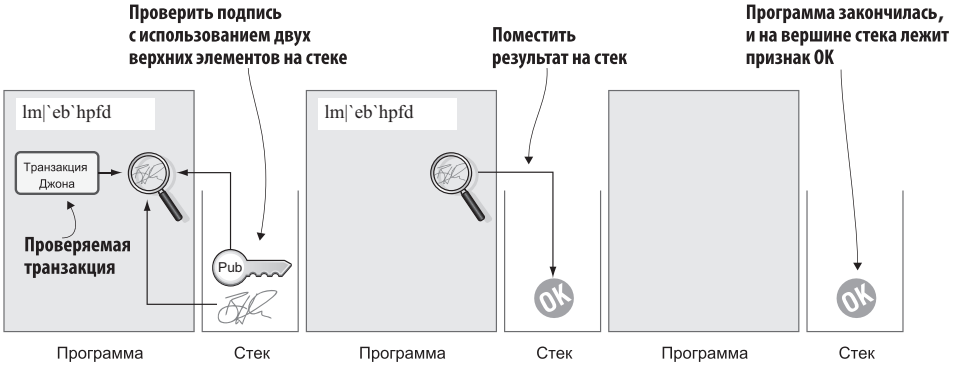


Рис. 5.15. Проверка подписи с использованием транзакции Джона и остальных элементов в стеке

Последний оператор в программе, `OP_CHECKSIG` (рис. 5.15), означает «Проверить, что открытый ключ на вершине стека и лежащая ниже подпись правильно подписывают транзакцию. Поместить `true` или `false` на вершину стека, в зависимости от результата проверки». Лиза берет транзакцию Джона и удаляет из нее сценарий подписи из всех входов. Затем использует два верхних элемента из стека — открытый ключ Джона и его подпись, — чтобы убедиться, что подпись соответствует транзакции. Когда Джон подписывал эту транзакцию, в ней не было подписей во входах. Вот почему Лиза должна сначала удалить сценарий подписи перед проверкой. Подпись определена как верная, поэтому Лиза вернула в стек значение `true`, то есть признак OK.

На этом программа заканчивается! Больше ничего не нужно делать. Верхний элемент на стеке, оставшийся после выполнения программы, показы-

вает, допустима ли передача средств в выход. Если на стеке лежит значение `true` — признак `OK`, — значит, операция расходования средств законна. Если на стеке лежит значение `false` — признак ошибки, — транзакция должна быть отклонена. Лиза смотрит на верхний элемент в стеке и видит признак `OK`. Теперь она знает, что вход с индексом 0 в транзакции Джона допустим (рис. 5.16).

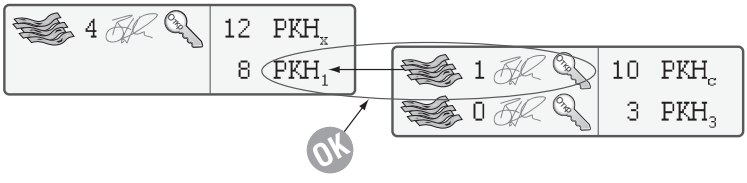


Рис. 5.16. Первый вход проверен

Затем Лиза выполняет аналогичную проверку для второго входа в транзакции Джона, с индексом 1. Если и эта программа завершится с признаком `OK`, значит, вся транзакция является действительной и ее можно добавить в электронную таблицу.

Почему выбрано решение с использованием программы?

Часть программы — сценарий открытого ключа — описывает, как именно должно оформляться расходование средств в транзакции. Единственный способ потратить средства — предоставить сценарий подписи, который завершается признаком OK на вершине стека.

В только что представленном примере сценарий подписи завершается с признаком успеха, только если за действительной подписью следует открытый ключ, соответствующий PKH в сценарии открытого ключа.

Использование языка программирования Script в транзакциях делает их очень гибкими. В этой книге вы увидите несколько разных видов программ на языке Script. Если бы в транзакциях не использовались программы, все варианты использования пришлось бы предусмотреть зара-

ОПЕРАТОРЫ

В программах на языке Script можно использовать множество самых разных операторов. Полный их список можно найти на веб-ресурсе 13 (приложение B).

нее. Язык Script позволяет людям придумывать новые варианты использования по своему усмотрению.

Я уже упоминал, что «платеж на РКН» — не единственный способ оплаты. В сценарии открытого ключа можно написать любую программу. Например, можно написать сценарий открытого ключа, завершающийся признаком `OK`, только если сценарий подписи возвращает два числа, сумма которых равна 10. Или можно написать программу, заканчивающуюся признаком `OK`, только если сценарий подписи содержит исходный образ хеша SHA256. Например:

```
OP_SHA256
334d016f755cd6dc58c53a86e183882f8ec14f52fb05345887c8a5edd42c87b7
OP_EQUAL
```

Это позволит потратить средства в выходе любому, кто знает исходные данные, для которых SHA256 возвращает хеш `334d016f...d42c87b7`. Как вы, наверное, помните из главы 2, этот хеш дает текстовая строка «Hello!». Допустим, ваш сценарий подписи имеет вид

```
Hello!
```

Выполните инструкции в программе и убедитесь, что она работает и что все сценарии подписи, не содержащие правильного исходного образа, возвращают признак ошибки.

Почему сценарии называются сценарием подписи и сценарием открытого ключа?

Возможно, вам интересно, почему сценарий в выходной части называется *сценарием открытого ключа*, даже при том что обычно он не содержит открытого ключа, а сценарий во входной части называется *сценарием подписи*, хотя он не содержит подписи.

Раньше сценарий открытого ключа в транзакциях Биткоин использовался для передачи фактического открытого ключа, а сценарий подписи — для передачи подписи. В то время все было намного проще. Типичный сценарий открытого ключа выглядел так:

```
<открытый ключ> OP_CHECKSIG
```

СТРАННЫЕ НАЗВАНИЯ

Разработчики Биткоин обычно используют термины *scriptPubKey* и *scriptSig* для обозначения сценариев открытого ключа и сценария подписи, потому что именно так они названы в исходном коде Bitcoin Core.

а сценарий подписи так:

```
<подпись>
```

С тех пор многое изменилось, но названия *сценарий подписи* и *сценарий открытого ключа* остались. Большинство разработчиков в наши дни смотрят на это более абстрактно: сценарий открытого ключа можно интерпретировать как открытый ключ, а сценарий подписи — как подпись, но не всегда они должны представлять обычные открытые ключи и подписи. В обычном современном платеже «открытый ключ» — это сценарий, который должен соответствовать «подписи» в сценарии подписи. Конечно, «открытый ключ» здесь содержит некоторые операторы и РКН, но на концептуальном уровне его можно считать открытым ключом. То же относится к сценарию подписи, который на концептуальном уровне можно считать подписью.

На чем мы остановились?

Эта глава охватывает основные аспекты транзакций. Рисунок 5.17 напоминает, как отправляется типичная транзакция.

Мы рассмотрели устройство транзакции и обсудили разные способы аутентификации, или «подписания», транзакций.

Рис. 5.17. Эта глава охватывает транзакции. Мы только что рассмотрели разные способы аутентификации транзакций

Необычные виды платежей

Транзакция Джона, которую мы только что рассмотрели, потратила два выхода, выполнив *платеж на хеш открытого ключа* (pay-to-public-key-hash, p2pkh). Но, как отмечалось выше, возможны другие виды платежей.

Например, платеж на хеш (pay-to-hash), когда средства переводятся на хеш SHA256. Чтобы потратить этот выход, вы должны передать в сценарии подписи исходный образ хеша. Далее мы рассмотрим еще более интересные и полезные способы аутентификации транзакций.

ПЛАТЕЖ НА ХЕШ

```
OP_SHA256
334d...87b7
OP_EQUAL
```

Несколько подписей

В случае платежа на хеш открытого ключа (p2pkh) получатель генерирует адрес и вручает его отправителю. Затем отправитель выполняет платеж на указанный адрес.

А теперь представим, что получатель захотел защитить деньги еще чем-то, кроме одного закрытого ключа. Предположим, Фаиза, Элен и Джон решили собрать деньги на благотворительность.

Они могут использовать обычный адрес p2pkh, на который их сторонники будут переводить жетоны на булочки. Они могли бы договориться позволить Фаизе управлять закрытым ключом, вследствие чего только она сможет потратить средства. У этого подхода есть несколько проблем:

1. Если Фаиза вдруг неожиданно исчезнет, деньги будут потеряны навсегда. Элен и Джон не смогут вернуть средства.
2. Если Фаиза с пренебрежением отнесется к резервному копированию, деньги могут быть потеряны. И никто не сможет восстановить их.
3. Если Фаиза с пренебрежением отнесется к безопасности закрытого ключа, деньги могут быть украдены.
4. Фаиза может сбежать с деньгами.

Как видите, этой схеме присуще множество рисков, но что, если Фаиза передаст закрытый ключ двум другим партнерам по благотворительности? Тогда все они смогут потратить деньги. Этот подход решает проблемы 1 и 2, но проблемы 3 и 4 становятся острее, потому что теперь любой из трех партнеров может проявить небрежность к безопасности закрытого ключа или сбежать с деньгами.

Эта группа организаторов состоит из трех человек. Было бы лучше, если бы они могли как-то *разделить ответственность и власть над деньгами*. Благодаря языку программирования Script это вполне возможно.

Каждый из них может создать свой закрытый ключ и потребовать, чтобы транзакция на расходование средств из благотворительного фонда была подписана двумя ключами из трех (рис. 5.18).

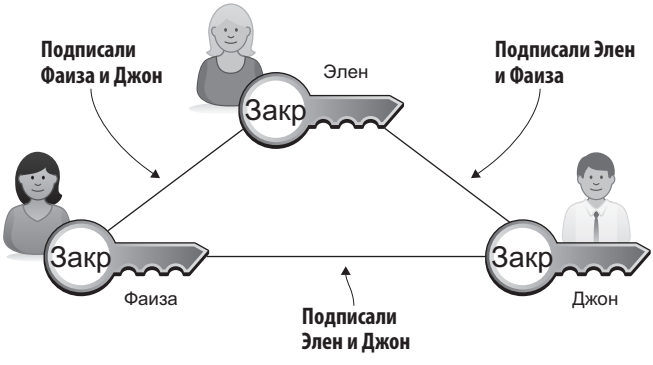


Рис. 5.18. Схема с несколькими подписями, в которой участвуют Фаиза, Элен и Джон. Чтобы потратить деньги, транзакция должна быть подписана двумя ключами из трех

Такой подход придает благотворительному счету ряд положительных свойств:

- Если один из трех ключей будет украден, вор не сможет украсть деньги.
- Если один из трех ключей будет потерян из-за пренебрежения резервным копированием или из-за исчезновения одного из организаторов, двух других ключей будет достаточно, чтобы потратить деньги.
- Никто из партнеров не сможет в одиночку скрыться с деньгами.

На рис. 5.19 показана программа, реализующая принцип «два из трех».

ОШИБКА

В программном обеспечении Биткоин кроется ошибка, из-за чего инструкция `OP_CHECKMULTISIG` требует наличия дополнительного фиктивного элемента перед сценарием подписи.

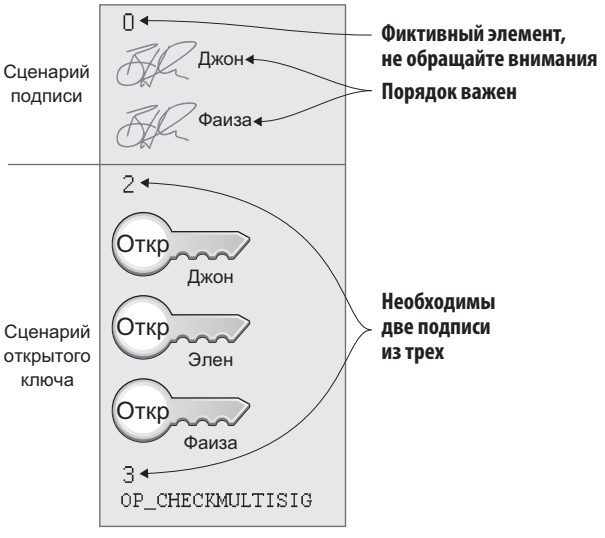


Рис. 5.19. Программа, требующая наличия двух подписей из трех. Суть кроется в инструкции OP_CHECKMULTISIG

Оператор `OP_CHECKMULTISIG` требует, чтобы Лиза убедилась, что две подписи в сценарии подписи созданы с использованием ключей в сценарии открытого ключа. Лиза выполняет инструкции из программы на рис. 5.20.

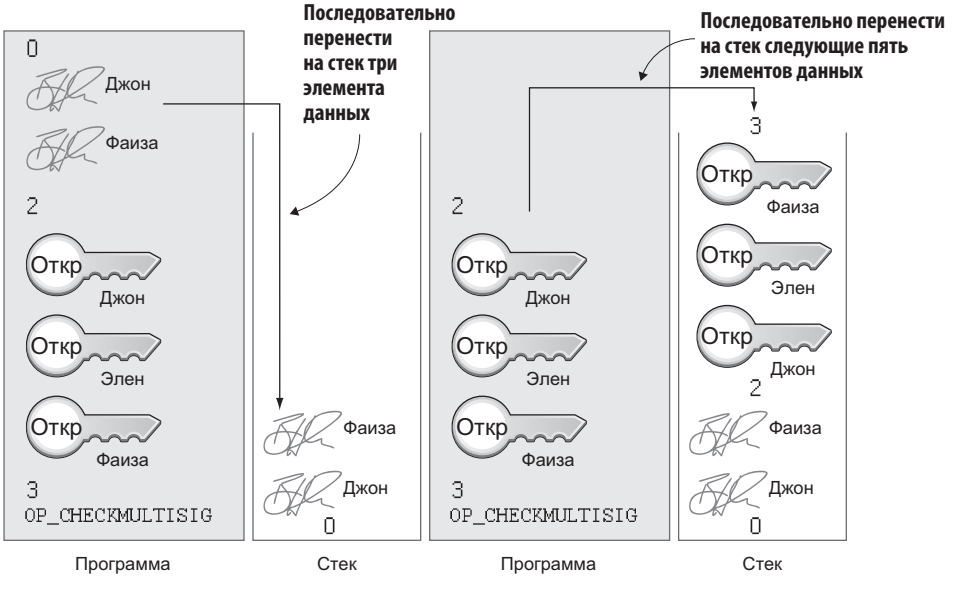


Рис. 5.20. Перемещение некоторых элементов данных на стек

Программа помещает на стек восемь элементов данных. Затем выполняется единственный оператор `OP_CHECKMULTISIG`, как показано на рис. 5.21. Оператор `OP_CHECKMULTISIG` снимает число со стека, в данном случае `3`, и интерпретирует его как число открытых ключей в стеке, за которым следует другое число. Это второе число определяет, сколько подписей необходимо, чтобы потратить деньги. В данном случае это число `2`. Затем оператор снимает со стека это число подписей, за которым следует фиктивный элемент, упомянутый выше. Он никак не используется в вычислениях.

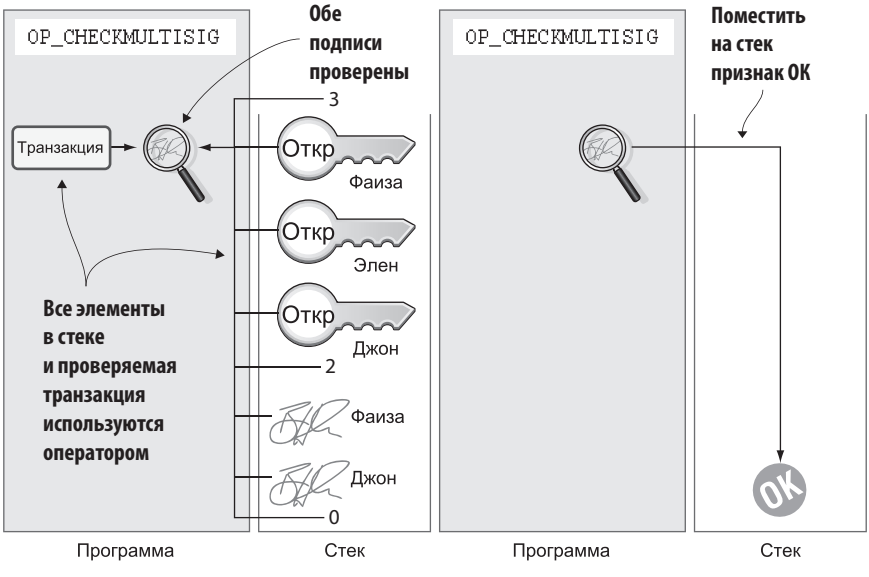


Рис. 5.21. Выполнение оператора OP_CHECKMULTISIG, который в данном случае возвращает признак OK

`OP_CHECKMULTISIG` использует всю эту информацию и транзакцию, чтобы убедиться в достаточном количестве подписей и их достоверности. Если все в порядке, он поместит на стек признак `OK`. На этом программа заканчивается. Поскольку по завершении программы на вершине стека находится признак `OK`, расходование средств разрешается.

Сотрудник, пожелавший пожертвовать жетоны на благотворительность, должен заставить свой кошелек записать сценарий открытого ключа, показанный

на рис. 5.19, в выход транзакции пожертвования. Это влечет за собой несколько проблем:

- Кошелек сотрудника может выполнять только платежи p2pkh. Его нужно модифицировать, чтобы он мог обрабатывать выходы с несколькими подписями и включать пользовательский интерфейс, чтобы сделать этот вид платежей понятным для пользователей.
- Обычно отправителю не обязательно знать, как защищены деньги получателя. Ему все равно, какого вида платеж он выполняет — с несколькими подписями, p2pkh или какой-то другой. Он просто хочет перевести средства.
- Часто за обработку транзакции взимается комиссия (подробнее об этом рассказывается в главе 7). Сумма комиссии обычно зависит от размера транзакции в байтах. Большой сценарий открытого ключа увеличивает сумму комиссии, которую должен заплатить отправитель. Это несправедливо, потому что такую необычную и дорогостоящую функцию хотел использовать получатель. Получатель, а не отправитель, должен платить за эту роскошь.

Вы можете исправить эту несправедливость, немого изменив порядок работы программы. Некоторые разработчики из числа сотрудников придумали нечто, получившее название *платеж на хеш сценария* (pay-to-script-hash, p2sh).

Платеж на хеш сценария

Мы уже знаем, как платеж вида p2pkh скрывает открытый ключ от отправителя за счет передачи ему хеша открытого ключа для оплаты вместо самого открытого ключа.

Разновидность p2sh развивает эту идею еще дальше — она скрывает программу. Вместо большого и сложного сценария открытого ключа получатель передает отправителю только хеш этого сценария. Затем отправитель осуществляет платеж на этот хеш и оставляет за получателем возможность представить сценарий позже, когда получатель захочет потратить деньги.

BIP16

Этот вид платежа был предложен в 2012 году в BIP16.

Снова предположим, что Фаиза, Элен и Джон решили собрать деньги на благотворительность и хотят защитить их множественной подписью (рис. 5.22).

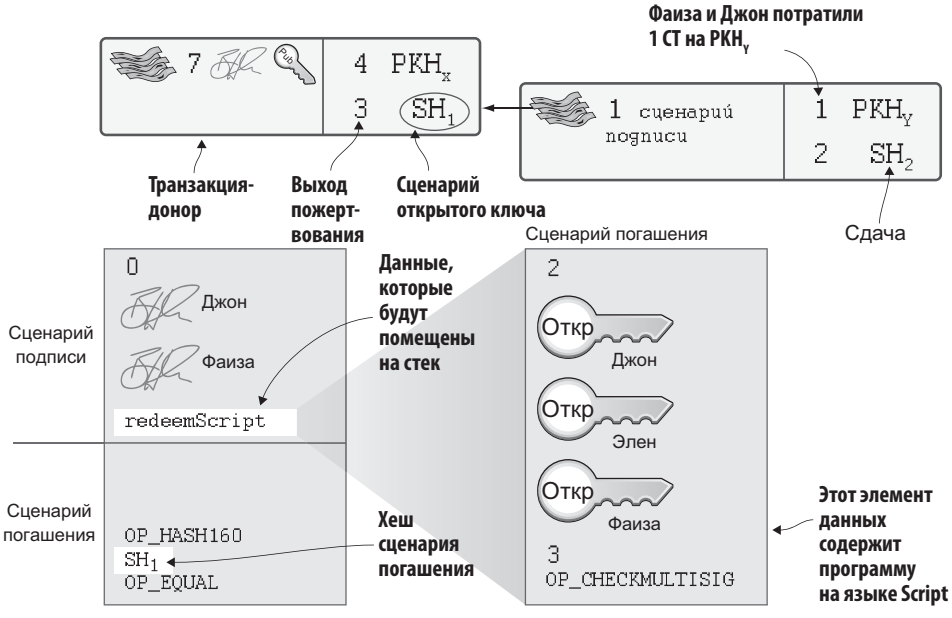


Рис. 5.22. Принцип работы платежа p2sh. Сценарий открытого ключа прост. Сценарий подписи, напротив, существенно сложнее, потому что содержит элемент данных с программой

Чтобы полностью проверить эту транзакцию, необходимо новое программное обеспечение. О том, как это новое программное обеспечение проверяет такие транзакции, мы поговорим чуть ниже. Но прежде посмотрим, как эту транзакцию обработает старое программное обеспечение.

Старое программное обеспечение

Что случится, если человек, проверяющий транзакцию, не обновит свое программное обеспечение до новейшей версии, поддерживающей проверку платежей p2sh? Разработчики постарались обеспечить прямую совместимость, то есть старое программное обеспечение не будет отклонять эти новые транзакции.

Представим, что в кафе используется старое программное обеспечение для проверки транзакций в электронной таблице (рис. 5.23). Старое программное обеспечение будет делать то же, что делало всегда — поместит на стек данные из сценария подписи и выполнит сценарий открытого ключа.

Когда программа завершится, на вершине стека останется элемент `true`, или `OK`. То есть старое программное обеспечение признает платеж действительным.

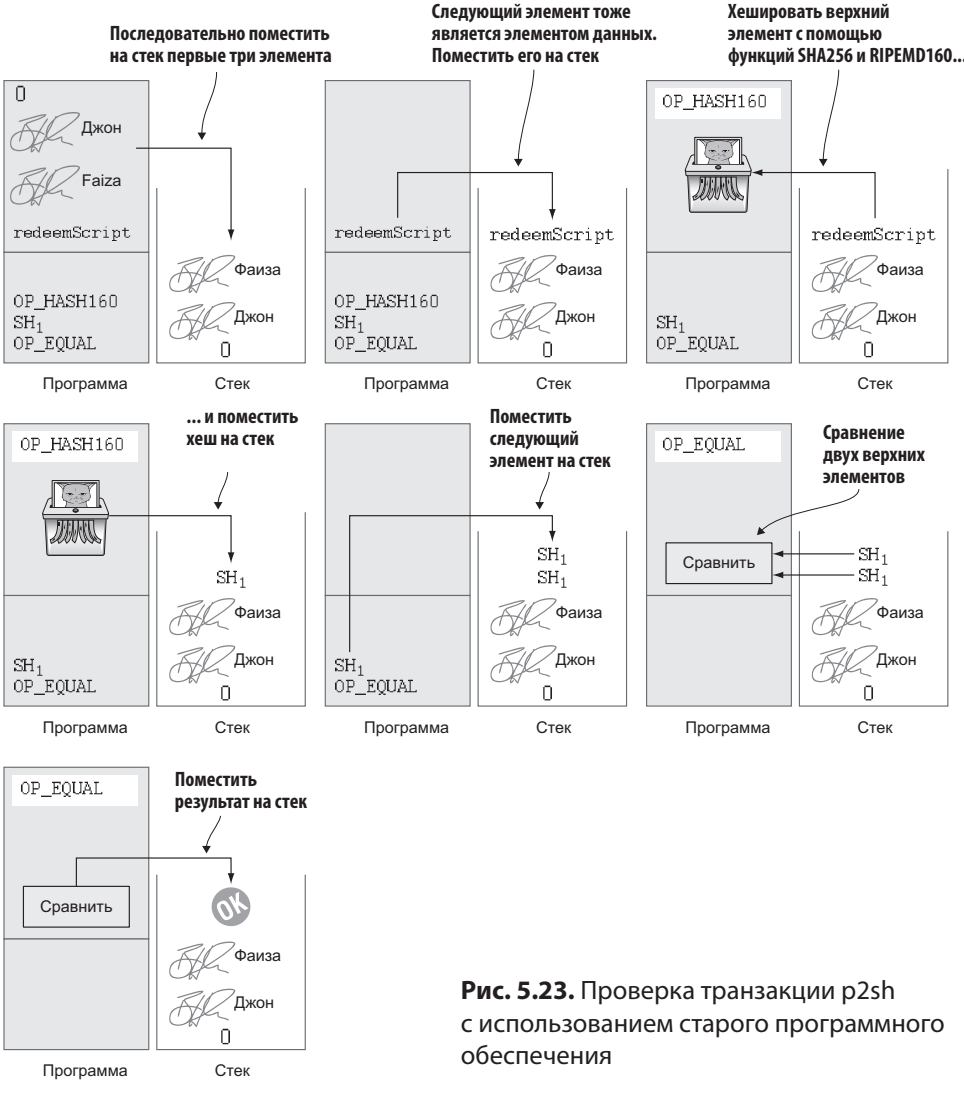


Рис. 5.23. Проверка транзакции p2sh с использованием старого программного обеспечения

Вы можете узнать сценарий открытого ключа из более раннего примера платежа на исходный образ хеша. Именно это здесь и произошло, но с другой криптографической хеш-функцией.

Старое программное обеспечение интерпретирует эту программу как платеж на хеш. Тот, кто представит исходный образ этого хеша, сможет получить деньги. Фактическая программа поддержки нескольких подписей, содержащаяся в сценарии погашения, никогда не запускается.

ЗАЧЕМ ПРОВЕРЯТЬ?

Кафе никак не вовлечено в эту транзакцию, так зачем им понадобилось бы проверять ее? В кафе хотят знать, насколько добросовестно Лиза относится к своей работе. Кафе заинтересовано в том, чтобы своевременно узнать, что происходит что-то подозрительное.

Новое программное обеспечение

Предположим, что в кафе только что обновили свое программное обеспечение и теперь хотят вновь проверить эту транзакцию. Посмотрим, как выполнится эта проверка.

Новое программное обеспечение обнаружит сценарий открытого ключа и по следующему шаблону определит, что эта транзакция описывает платеж вида p2sh:

```
OP_HASH160
20 байт хеша
OP_EQUAL
```

Если сценарий открытого ключа в точности соответствует этому шаблону — шаблону p2sh, — программное обеспечение будет обрабатывать программу иначе. Сначала будут выполнены те же семь шагов, что и в старом программном обеспечении, показанные на рис. 5.23, но программа сохранит стек после шага 2. Назовем его *сохраненным стеком*. Если первые семь шагов дадут в результате признак `OK`, стек заменяется сохраненным стеком, и верхний элемент — сценарий погашения `redeemScript` — удаляется из стека (рис. 5.24).

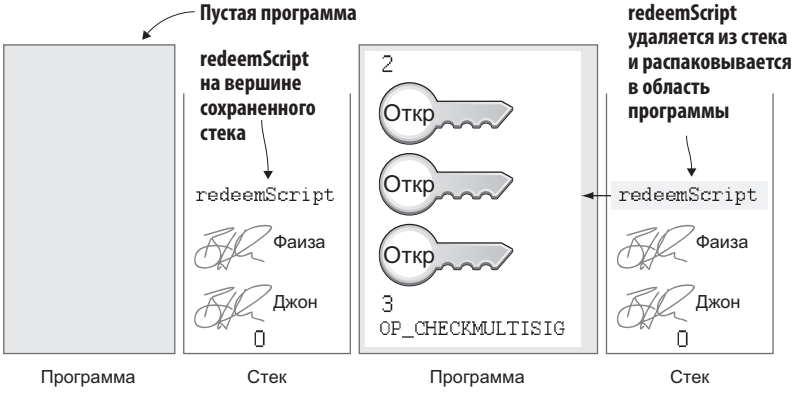


Рис. 5.24. Стек замещается сохраненным стеком, и сценарий погашения redeemScript удаляется из стека

`redeemScript` — это элемент данных, содержащий программу, как уже говорилось выше. Теперь эта программа переносится в область программы и запускается на выполнение. С этого момента она действует, как если бы выполнялся платеж в старом стиле (рис. 5.25).

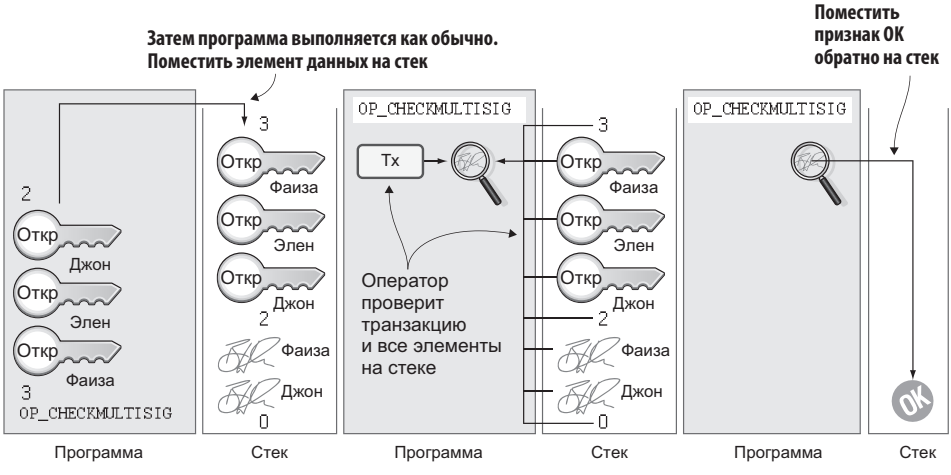


Рис. 5.25. Выполнение программы, содержащейся в сценарии погашения

Для Лизы важно использовать самое новое программное обеспечение. Если Лиза будет пользоваться старым программным обеспечением, она сможет проверить только совпадение хеша сценария погашения с хешем в сценарии открытого ключа. Любой, в чьи руки случайно попал сценарий погашения, например Фаиза, сможет завладеть деньгами в таблице. Лиза благополучно подтвердит эту транзакцию. Это может вызвать проблемы, если какие-то проверяющие узлы используют новое программное обеспечение. В соответствии с новыми правилами эти узлы признают транзакцию в электронной таблице недействительной. С этого момента вся электронная таблица станет недействительной и неприемлемой для новых узлов. Подробнее эту ситуацию мы рассмотрим в главе 11.

Адреса для платежей на хеш сценария

Фаиза, Элен и Джон создали свой сценарий погашения, требующий двух подписей из трех:

```
2
022f52f2868dfc7ba9f17d2ee3ea2669f1fea7aea3df6d0cb7e31ea1df284bdaec
023d01ba1b7a1a2b84fc0f45a8a3a36cc7440500f99c797f084f966444db7baeee
```

```
02b0c907f0876485798fc1a8e15e9ddabae0858b49236ab3b1330f2cbadf854ee8
3
OP_CHECKMULTISIG
```

Теперь они хотят, чтобы люди проводили платежи на хеш SHA256+ RIPEMD160 этого сценария:

```
04e214163b3b927c3d2058171dd66ff6780f8708
```

Как Фаизе, Элен и Джону все организовать? Что они должны напечатать на листовках, чтобы коллеги могли перевести благотворительные взносы на хеш сценария? Давайте рассмотрим несколько доступных вариантов:

- Напечатать хеш сценария как есть и сообщить коллегам, что это хеш сценария погашения. В этом случае велик риск допустить опечатку, так же как в случае с платежами на обычные PKH, которые обсуждались в главе 3.
- Вычислить хеш base58check сценария, как в главе 3, и получить адрес вида `1SpXyW...RMmEMZ`. Этот адрес можно напечатать на листовках, раздать коллегам и сообщить, что они должны выполнить платеж типа p2sh вместо обычного p2pkh.

В обоих случаях, если благотворитель ошибочно выполнит платеж типа p2pkh, используя напечатанный хеш или адрес, никто не сможет потратить деньги, потому что этому ложному PKH не соответствует ни один закрытый ключ.

Эти два варианта не выглядят ни безопасными, ни практичными. Поэтому познакомимся с новым форматом адресов для p2sh, *адресом p2sh* (рис. 5.26). Этот формат похож на обычные адреса p2pkh. Он использует схему кодирования base58check, как и обычные адреса.

Эта процедура почти в точности повторяет процедуру создания адресов p2pkh. Единственное отличие в номере версии — `05` вместо `00`. В результате получается адрес, начинающийся с `3` вместо `1`.

Из-за этого изменения и особенности работы алгоритма base58, связанной с использованием целочисленного деления на 58, последний остаток всегда будет равен 2. Если вам интересно, на рис. 5.27 подробно показано, как кодируется хеш сценария погашения Фаизы, Элен и Джона с номером версии и контрольной суммой.

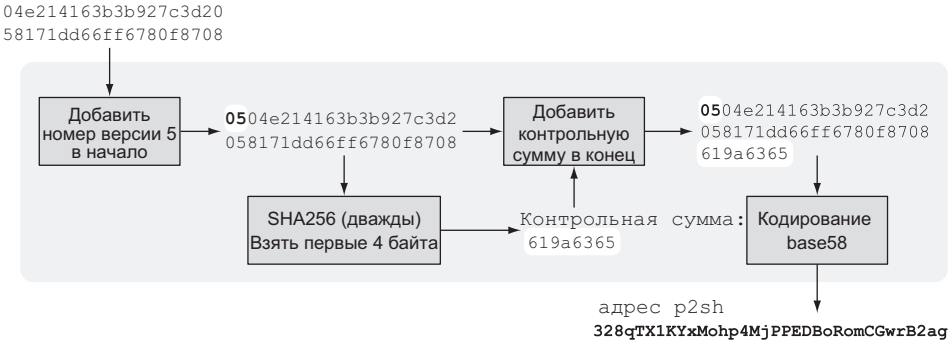


Рис. 5.26. Создание адреса p2sh. От обычных адресов он отличается номером версии: 05 вместо 00

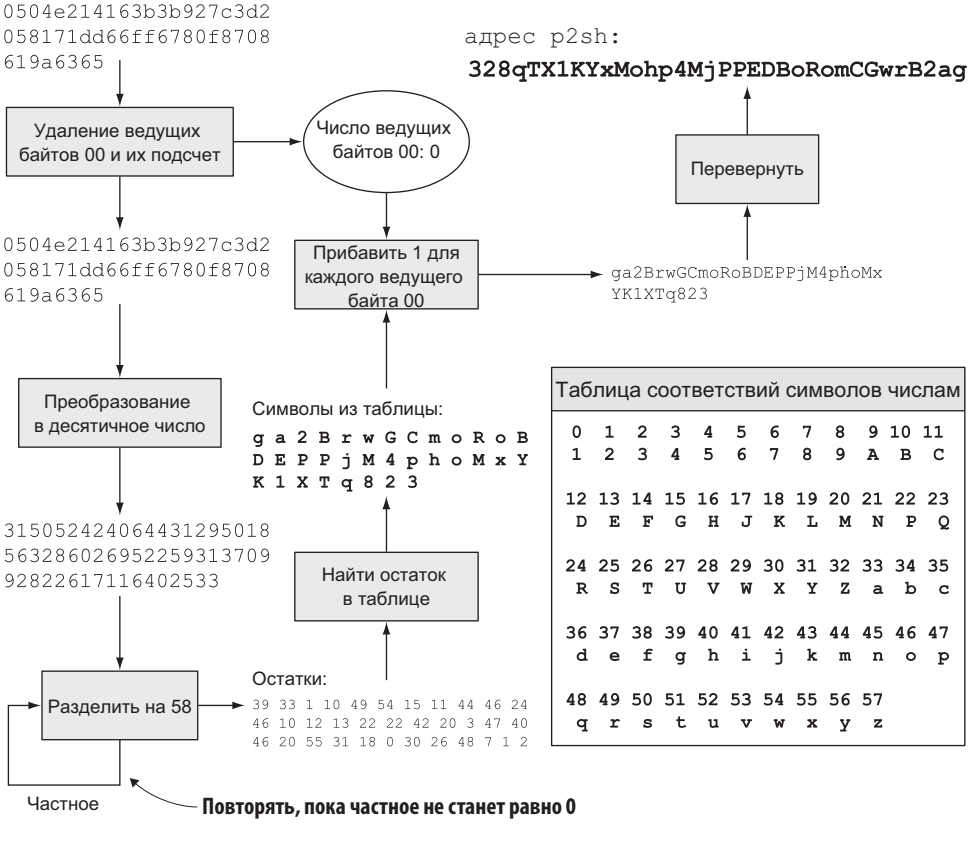


Рис. 5.27. Кодирование хеша сценария с номером версии и контрольной суммой с использованием функции base58. Результат всегда начинается с символа 3

Этот последний остаток 2 будет преобразован функцией base58 в 3 в результате поиска по таблице символов. Этот символ 3 станет первым, когда процедура кодирования перевернет полученный хеш. По этой причине все адреса p2sh начинаются с 3. Именно так пользователи отличают адреса p2sh от других адресов, например p2pkh.

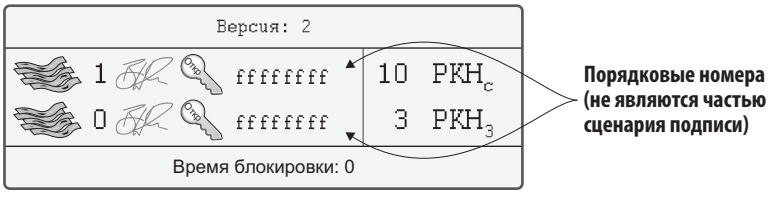
Фаиза, Элен и Джон теперь могут печатать `328qTX...wrB2ag` на своей листовке. Когда кто-то из сотрудников отсканирует QR-код на листовке, их кошелек распознает адрес как адрес p2sh, потому что он начинается с 3. Кошелек выполнит base58check-декодирование адреса и создаст правильный выход p2sh в транзакции:

```
OP_HASH160
04e214163b3b927c3d2058171dd66ff6780f8708
OP_EQUAL
```

На этом мы завершаем обсуждение темы программирования транзакций. Теперь вы знаете, что транзакции могут выражать множество различных правил, регламентирующих расходование денег. Обратите внимание: нельзя ограничить, куда уйдут потраченные деньги, можно только определить, что необходимо, чтобы их потратить. Сценарий открытого ключа определяет, что должно присутствовать в сценарии подписи. Далее в этой книге мы снова вернемся к транзакциям и поговорим о еще более интересных возможностях, которые они предлагают, например о том, как сделать невозможным расходование средств до определенной даты в будущем.

Дополнительные элементы в транзакциях

Мы описали только часть содержимого транзакции. В них присутствует еще кое-какая информация, включая номер версии, время блокировки и порядковые номера.



- *Версия* — каждая транзакция имеет версию. На момент написания этих строк существовало две версии: 1 и 2.
- *Порядковый номер* — 4-байтное число в каждом входе. В большинстве транзакций устанавливается максимально возможное значение `ffffffff`. Это старая и неподдерживаемая особенность, предназначенная для новых возможностей.
- *Время блокировки* — отметка времени, до которой транзакция не может быть добавлена в электронную таблицу. Если время блокировки равно 0, транзакцию можно добавить в электронную таблицу в любой момент.

Я перечислил эти редкие элементы здесь только для полноты. Мы обсудим их подробнее в главе 9, когда вы узнаете больше об основах Биткоин.

Вознаграждение и создание монет

Возможно, вам интересно узнать, откуда берутся все эти жетоны на булочки. Вспомните, как в главе 2 я писал, что Лиза получает 7200 CT в день. Каждый день она добавляет в таблицу новую запись, выплачивая себе 7200 новых CT.

От кого	Кому	Количество CT
...	...	...
Кафе	Компания	10,000
Алиса	Кафе	10
НОВЫЕ	Лиза	7200

Она все так же вознаграждает себя 7200 CT в день, но теперь немного по-другому. Каждый день она добавляет в электронную таблицу специальную транзакцию, которая называется *исходной монетарной транзакцией* (*coinbase transaction*), как показано на рис. 5.28.

Рис. 5.28. Лиза каждый день вознаграждает себя, добавляя исходную монетарную транзакцию

Вход исходной монетарной транзакции называется *coinbase*. Создать новые монеты можно, только добавив в электронную таблицу исходную монетарную транзакцию. Новые монеты создаются как вознаграждение Лизы за выполнение ценной работы.

Любую транзакцию можно проследить в обратном порядке до одной или нескольких исходных монетарных транзакций, следуя по идентификаторам транзакций во входах. Транзакции образуют *граф транзакций* (рис. 5.29). Они связаны посредством идентификаторов транзакций.

ВОЗНАГРАЖДЕНИЕ

Вознаграждения в системе Биткоин выплачиваются узлам, защищающим блокчейн Биткоин, примерно каждые 10 минут с использованием исходных монетарных транзакций. Подробнее об этом я расскажу в главе 7.

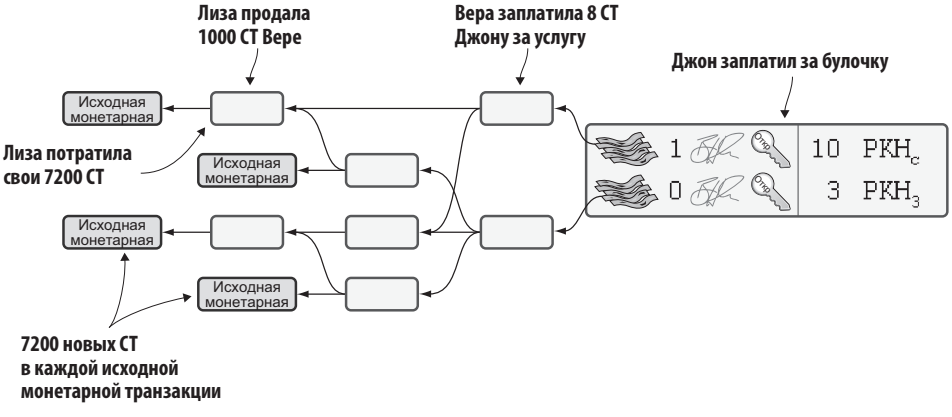


Рис. 5.29. Граф транзакций. Все транзакции берут начало из одной или нескольких исходных монетарных транзакций

Транзакция Джона уходит корнями в четыре разные исходные монетарные транзакции. Чтобы проверить транзакцию Джона, нужно проследовать по всем идентификаторам транзакций, начиная с транзакции Джона, и проверить все транзакции, встречающиеся на пути, пока не будут достигнуты четыре монетарные транзакции. В такого рода проверках поможет множество UTXO. В нем хранятся все уже проверенные UTXO. Проверяющим останется только проследовать по идентификаторам транзакций (обычно требуется сделать всего один шаг), пока они не достигнут выхода, присутствующего в множестве UTXO.

Исходные монетарные транзакции тоже должны проверяться — в каждые сутки должна иметься только одна монетарная транзакция, и каждая из них должна создавать ровно 7200 новых жетонов на булочки.

Переход от версии 4.0

Возможно, вам интересно, как был выполнен переход от старой электронной таблицы — имевшейся в версии 4.0 системы — к новой, содержащей транзакции. Что случилось с жетонами, уже имеющимися в электронной таблице?

Сотрудники заранее договорились о временном интервале, когда произойдет такой переход. В течение этого времени Лиза создала одну огромную транзакцию, содержащую по одному выходу для каждого PKH в электронной таблице. Эта транзакция выглядит как исходная монетарная транзакция, но имеет большое количество выходов. Любой может сохранить версию старой электронной таблицы и убедиться, что эта новая транзакция содержит те же выходы, что и старое множество UTXO. Новые проверяющие не могут быть уверены, что все прошло без сучка и задоринки, поэтому им придется довериться Лизе.

Обратите внимание, что эта ситуация отличается от ситуации в системе Биткоин, которая изначально использовала транзакции. В «начале всех начал» в Биткоин имелось пустое множество UTXO. И ни у кого не было биткоинов.

Доверие к Лизе

В этой главе мы формализовали процесс оплаты — например, кошелек должен был отправлять транзакцию Лизе в виде вложения в электронное письмо. Лиза смогла воспользоваться этим формальным процессом и автоматизировать свою работу. Она написала компьютерную программу, которая автоматически читает транзакции из ее почтового ящика и проверяет их, поддерживая множество UTXO и добавляя транзакции в электронную таблицу. Лиза может расслабиться и просто наблюдать, как ее программа делает всю работу за нее. Чудесно.

Но теперь возникает вопрос, стоят ли ее услуги 7200 CT в день. Она больше не занимается проверкой, а просто сидит сложа руки. Давайте задумаемся, за что мы ее вознаграждаем. Она вознаграждается не за скучную ручную

работу, а за правильное и честное подтверждение транзакций. Именно это представляет ценность для вас и ваших коллег. Лиза написала компьютерную программу, на которую взвалила всю тяжелую работу, но это не делает обработку платежей менее правильной или честной.

Транзакции решают проблему возможности произвольного изменения Лизой содержимого в электронной таблице. Единственное, в чем мы еще вынуждены доверять Лизе, это:

- *отсутствие цензуры транзакций* — она обязана добавлять в таблицу любые допустимые транзакции, полученные по электронной почте;
- *невозможность отмены транзакций* — отмена транзакции означает ее удаление из электронной таблицы.

МЫ ВЕРИМ, ЧТО ЛИЗА НЕ...

- цензурирует транзакции;
- отменяет транзакции.

Если Лиза невзлюбит Фаизу и ей известны некоторые UTXO, принадлежащие Фаизе, она может отказаться обрабатывать транзакции Фаизы, расходующие эти UTXO. Это означает, что Фаиза не сможет потратить свои деньги. Лиза подвергает цензуре транзакции Фаизы.

Если Лиза удалит из электронной таблицы транзакцию, выходы которой еще не были израсходованы, это *может* быть замечено проверяющими, занимающимися проверкой уже некоторое время. Но новые проверяющие, приступившие к работе после отмены транзакции, не заметят этого, потому что электронная таблица все еще соответствует правилам.

Допустим, Лиза отменила транзакцию Джона, которую мы описали в разделе «Платежи с использованием транзакций». Лиза удаляет транзакцию Джона из таблицы. Никто еще не успел потратить ни одного из выходов в транзакции Джона, поэтому в электронной таблице нет транзакций, которые станут недействительными после удаления транзакции Джона.

Проверяющие, занимающиеся проверкой уже некоторое время — например сотрудники кафе, — не заметят этого, потому что они просматривают в электронной таблице только последние добавленные транзакции. Они уже проверили транзакцию Джона и добавили ее в свои множества UTXO. Сотрудники кафе верят, что Лиза не будет удалять транзакции, поэтому они никогда не выполняют повторных проверок электронной таблицы.

Предположим также, что новый сотрудник, Вера, начала создавать свое множество UTXO из электронной таблицы, в которой сейчас отсутствует

транзакция Джона. Это множество UTXO будет отличаться от множества UTXO в кафе. С точки зрения Веры, Джон все еще имеет деньги, и он не заплатил 10 CT кафе. Выходы, которые Джон потратил в своей транзакции, кажутся Вере неизрасходованными, потому что они находятся в множестве UTXO Веры.

Теперь у нас есть Вера, которая думает, что у Джона еще есть деньги; Лиза, удалившая транзакцию; и кафе, сотрудники которого думают, что получили 10 CT от Джона. Пока никто не заметил проступка Лизы. Он останется незамеченным, пока никто не попытается потратить выходы транзакции Джона, например в кафе решат потратить свои 10 CT или Джон решит израсходовать свою сдачу в 3 CT.

Допустим, в кафе решили заплатить арендную плату компании. Среди прочих в платеже будут указаны выходы транзакции Джона. Кафе создает транзакцию, которая расходует выходы, подписывает их и отправляет Лизе. Лиза помнит, что она удалила транзакцию Джона, и теперь ее проступок может вскрыться. Если Лиза решит подтвердить транзакцию кафе, она сделает всю электронную таблицу недействительной, и тогда Вера и все другие проверяющие отклонят электронную таблицу целиком. Это очень плохо. Если Лиза решит отклонить транзакцию, что для нее было бы более разумно, в кафе заметят это, потому что их транзакция останется неподтвержденной.

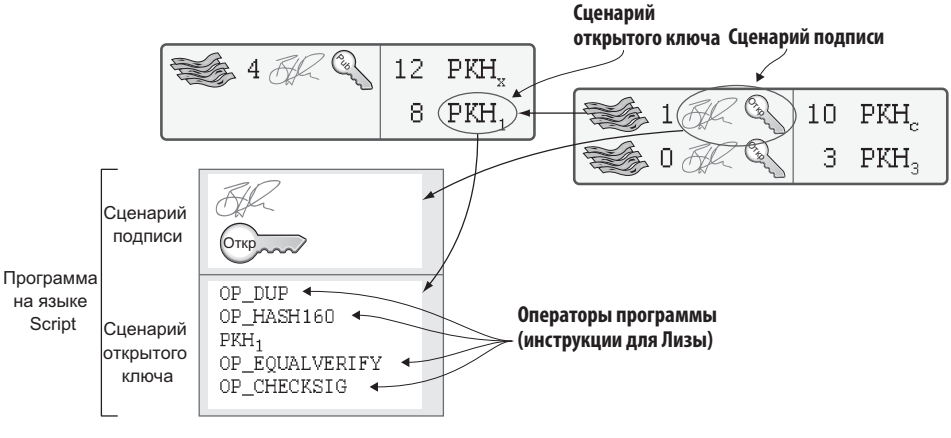
Когда в кафе заметят это, они не смогут доказать, что транзакция Джона когда-либо присутствовала в электронной таблице. Лиза не сможет доказать, что транзакции Джона никогда не было в электронной таблице. Получается слово против слова. Мы решим эту проблему в главе 6.

Непонятно, почему Лиза удалила транзакцию Джона. Возможно, Джон заплатил Лизе за это. Возможно, у Лизы были свои мотивы. Представьте, что она покупает булочку в кафе, и когда в кафе увидели в электронной таблице транзакцию от Лизы на свой адрес, ей выдали булочку. Вкуснятина. Затем Лиза возвращается к своему столу и отменяет транзакцию. Теперь у нее есть булочка, и она сохранила деньги. Это, конечно, будет замечено, когда кафе попытается потратить выходы из удаленной транзакции, или в следующий раз, когда Лиза попытается вторично потратить деньги из удаленной транзакции. Но, как и в случае с Джоном, это слово против слова. Лиза может утверждать, что транзакции никогда не было в таблице, и сотрудники кафе могут утверждать, что она была. Но никто не сможет что-либо доказать.

Повторение

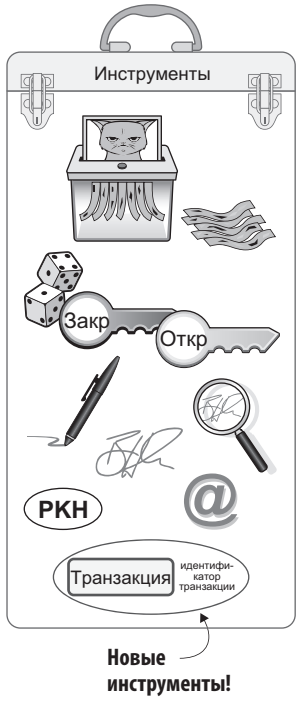
Транзакции не позволят Лизе украсть жетоны у других. Они решают эту проблему, делая общедоступными все подписи в электронной таблице. Кошельки пользователей создают и подписывают транзакции, которые Лиза проверяет и добавляет в электронную таблицу.

Транзакции имеют входы и выходы. Выход транзакции содержит последнюю часть программы на языке Script. Вход транзакции, расходующей этот выход, должен включать первую часть программы.



Лиза выполняет программу. Если она завершится с признаком `OK`, значит, расходование *этого* выхода разрешено. Если программы во всех входах в транзакции вернут признак `OK`, вся транзакция будет признана действительной и Лиза добавит ее в электронную таблицу.

Когда транзакция окажется в электронной таблице, любой сможет выполнить те же проверки, что и Лиза, потому что транзакция добавляется в электронную таблицу именно в том виде, в каком была получена Лизой. Если Лиза внесет в нее изменения, люди заметят, что электронная таблица стала недействительной, поскольку содержит недопустимую транзакцию. Единственное, что нельзя проверить, — это цензуру транзакции (когда она не добавляется в электронную таблицу) или удаление из электронной таблицы. На данный момент нам приходится верить Лизе в этих двух аспектах.



Изменения в системе

В наш набор инструментов добавились транзакции и их идентификаторы. Таблица понятий (табл. 5.1) сократилась на две строки: из нее исчезли письмо Лизе и на смену записям в электронной таблице пришли транзакции. Обратите внимание, что мы все еще используем электронную почту для отправки транзакций Лизе, но сами транзакции имеют тот же формат, что и в системе Биткоин. Вот почему мы можем избавиться от понятия «запись».

Таблица 5.1. Транзакции заменили письма Лизе и записи в электронной таблице

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
Электронная таблица	Блокчейн	Глава 6
~~Письмо Лизе~~	~~Транзакция~~	~~Глава 5~~
~~Запись в электронной таблице~~	~~Транзакция~~	~~Глава 5~~
Лиза	Майнер	Глава 7

В следующей главе мы позаботимся о замене электронной таблицы, в которой сейчас содержатся транзакции, цепочкой блоков — блокчейном.

А теперь выпустим версию 5.0 системы жетонов на булочки (табл. 5.2).

Таблица 5.2. Примечания к релизу, жетоны на булочки 5.0

Версия	Особенность	Как реализована
НОВАЯ 5.0	Расходование нескольких «монет» в одном платеже	Несколько входов в транзакции
	Любой может проверить электронную таблицу	Подписи в транзакциях доступны всем
	Отправитель может определять критерии расходования денег	Программы на языке Script внутри транзакции
4.0	Простота платежей и создания новых адресов	Мобильное приложение «кошелек»
	Упрощенное резервное копирование	HD-кошельки генерируют ключи из начального случайного числа. Чтобы создать резервную копию, достаточно сохранить от 12 до 24 английских слов
	Создание адресов в небезопасном окружении	HD-кошельки способны генерировать деревья открытых ключей без использования закрытых ключей
3.0	Защищенность от дорогостоящих опечаток	Адреса в системе жетонов на булочки
	Улучшенная конфиденциальность	Вместо имен в электронной таблице сохраняются хеши открытых ключей (PKH)

Упражнения

Для разминки

5.1. Представьте, что все ваши деньги распределены по трем UTXO: один с 4 CT, один с 7 CT и один с 2 CT. Какой из этих выходов вы бы потратили, если бы решили купить булочку за 10 CT? Какие выходы будет иметь ваша транзакция и какие суммы в CT будут в них указаны?

5.2. Какой цели служат идентификаторы в транзакциях?

5.3. Почему часто требуется добавлять в транзакцию выход со сдачей?

5.4. Где в транзакции находится подпись?

5.5. Зачем нужен открытый ключ во входе транзакции, расходующей выход p2pkh?

5.6. Почему перед подписанием транзакции кошелек убирает из нее сценарии подписи?

5.7. Где в транзакции находятся сценарии открытого ключа и что они содержат?

5.8. Что требуется от программы на языке Script (сценарий подписи + сценарий открытого ключа), чтобы вход был признан аутентичным?

5.9. Какие отличительные черты имеет адрес p2sh?

Придется пораскинуть мозгами

5.10. Допустим, у вас есть 100 CT в одном выходе транзакции с индексом 7. Вы хотите перевести кафе 10 CT по p2pkh-адресу $@_C$ и 40 CT по адресу $@_{FEJ}$ благотворительного фонда, организованного Фаизой, Элен и Джоном. Сконструируйте единственную транзакцию, выполняющую эти платежи. Можете смело подглядывать в текст этой главы, чтобы отыскать нужные операторы и шаблоны программ. Вам не нужно подписывать какие-либо входы.

5.11. Множество UTXO содержит все UTXO. Предположим, что множество включает 10 000 UTXO и вы посылаете Лизе транзакцию с двумя входами и пятью выходами. Сколько UTXO будет содержать множество UTXO после подтверждения транзакции?

5.12. Создайте простой сценарий открытого ключа, который позволит любому потратить выход транзакции. Что должен содержать сценарий подписи во входе транзакции, расходующей эти средства?

5.13. Создайте сценарий открытого ключа, который потребует от того, кто будет расходовать средства, указать два числа в сценарии подписи, сумма которых равна 10. В языке Script есть оператор `OP_ADD`, который снимает со стека два верхних элемента и возвращает их сумму.

5.14. Представьте, что вы запустили полный узел и получили деньги от Фаизы в подтвержденной транзакции. Можете ли вы поверить в реальность этих денег?

5.15. Открытый ключ доступен во входе транзакции, расходующей выход p2pkh. В чем недостаток этого способа в случае, когда имеется несколько UTXO для одного PKH? Что можно сделать, чтобы избавиться от этого недостатка?

Итоги

- Транзакции имеют входы и выходы, поэтому есть возможность потратить несколько «монет» и заплатить нескольким получателям в одной транзакции.
- Выходы транзакций являются «программируемыми». Кошелек отправителя решает, какую программу поместить в выход. Он определяет, что необходимо, чтобы потратить деньги.
- Любой может проверить всю электронную таблицу, потому что все подписи являются общедоступными. Это здорово уменьшает необходимость верить Лизе.
- Сценарии можно использовать, чтобы потребовать наличия нескольких подписей, например трех из семи. Это отлично подходит для компаний и благотворительных организаций.
- Для многих необычных способов оплаты, как, например, с требованием наличия нескольких подписей, используется новый тип адреса — p2sh, начинающийся с 3.
- Все транзакции происходят от одной или нескольких монетарных транзакций. Монетарные транзакции — единственный способ создания денег.
- Любой сотрудник в компании может проверить создание денег, чтобы убедиться, что Лиза создает ровно столько, сколько было согласовано: 7200 CT в день, с уменьшением вдвое каждые четыре года.
- Лиза может цензурировать и отменять транзакции. Мы все еще вынуждены верить, что она не пойдет на это.

6 Блокчейн

Эта глава охватывает следующие темы:

- ✓ улучшение защищенности электронной таблицы;
- ✓ легкие кошельки;
- ✓ снижение требований к пропускной способности кошельков.

В главе 5 мы обсудили транзакции, которые позволяют любому проверить все сделки, зафиксированные в электронной таблице. Но осталось еще кое-что, чего не могут проверяющие: удаление и цензурирование транзакций Лизой. Как противостоять цензуре, мы обсудим в главах 7 и 8, а в этой главе посмотрим, как лишить Лизу возможности тайком удалять или подменять транзакции.

С этой целью мы заменим электронную таблицу цепочкой блоков — *блокчейном* (рис. 6.1). Блокчейн содержит транзакции, защищенные от подмены путем хеширования и подписания набора транзакций остроумным способом. Этот способ позволяет с легкостью представить криптографическое подтверждение мошенничества, если Лиза удалит или подменит транзакцию. Все проверяющие хранят свои копии блокчейна и могут выполнить его полную проверку, чтобы убедиться, что Лиза не удалила уже подтвержденные транзакции.

В этой главе также представлен легкий кошелек, или кошелек с *упрощенной проверкой платежей* (Simplified Payment Verification, SPV), который будет доверять проверку блокчейна кому-то другому — полному узлу — для экономии пропускной способности и места в памяти. Блокчейн предлагает такую возможность, но она имеет свою цену.

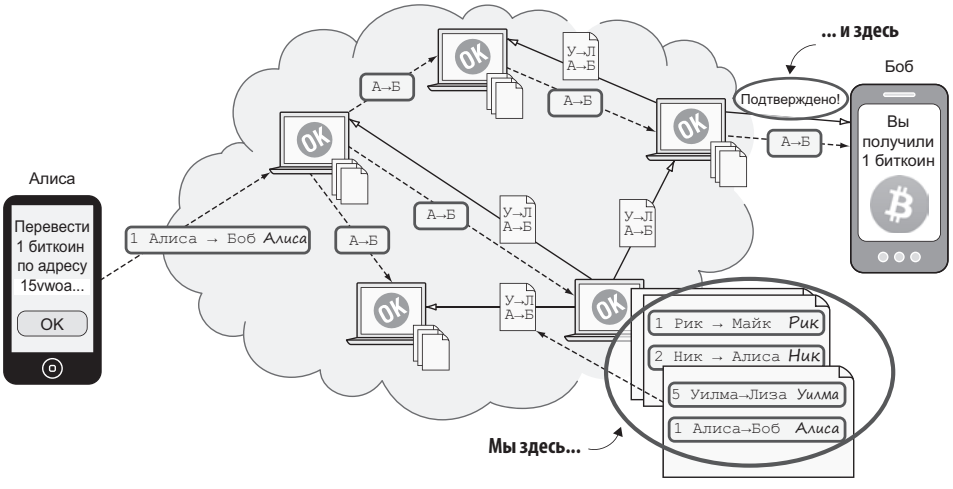


Рис. 6.1. Блокчейн в Биткоин

Лиза может удалять транзакции

Как уже не раз отмечалось, Лиза может удалять транзакции. Например, она может купить булочку в кафе, съесть ее и удалить транзакцию. Конечно, Лиза никогда так не поступит, потому что она самый надежный человек в мире, но не все ее коллеги знают или верят в это. Предположим, что она действительно удалила транзакцию, как показано на рис. 6.2.

Позже, когда в кафе заметят пропажу транзакции, они не смогут доказать, что транзакция Лизы когда-либо присутствовала в электронной таблице. И Лиза не сможет доказать, что ее там не было. Это весьма неприятная ситуация. Разбирательства, когда слово выступает против слова, могут оказаться длительными и дорогостоящими, возможно, с участием адвокатов, полиции, страховых компаний и следователей.

Можно ли доказать, что транзакция была в свое время подтверждена? Лизе нужен какой-то способ, позволяющий опубликовать транзакции и их порядок, чтобы исключить возможность их изменения.

Построение блокчейна

Возможность удаления транзакций обусловлена тем, что никто не сможет доказать, что список транзакций изменялся. Можно ли как-то изменить систему, чтобы получить возможность доказать, что Лиза вмешивалась в историю транзакций?

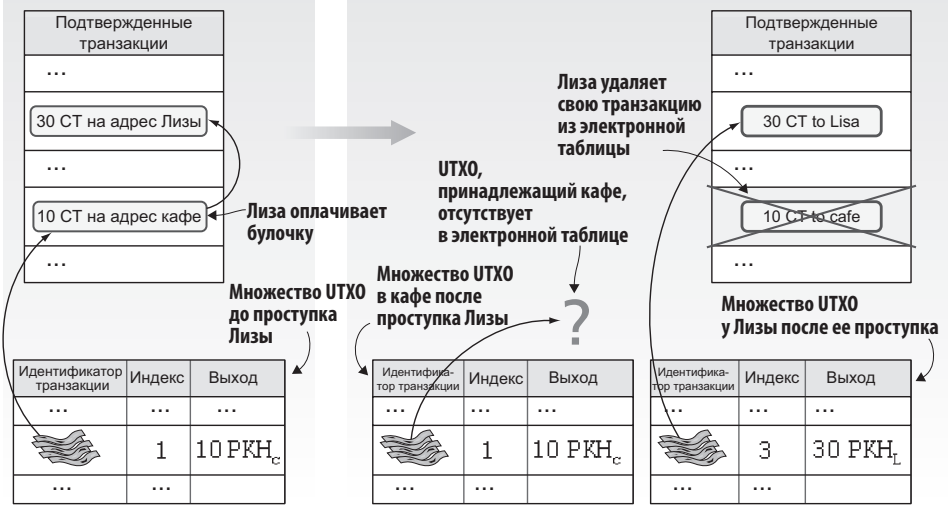


Рис. 6.2. Лиза купила булочку и отменила свою транзакцию. Фактически она украла булочку! В кафе и у Лизы теперь разное множество UTXO

Среди сотрудников нашлись разработчики, предложившие избавиться от электронной таблицы и заменить ее цепочкой блоков — блокчейном (рис. 6.3).

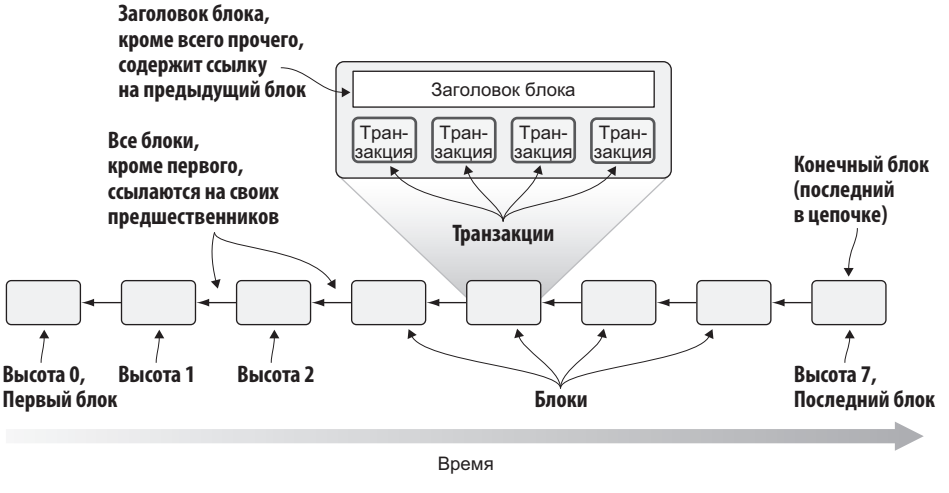


Рис. 6.3. Блокчейн — это цепочка блоков. Блоки содержат группы транзакций, и каждый из них ссылается на предыдущий

Каждый блок в блокчейне ссылается на предыдущий блок и имеет неявную *высоту*, которая указывает, насколько далеко он находится от первого блока. Первый блок имеет высоту 0, второй — высоту 1 и т. д. На рис. 6.3 *конечный*, или последний, блок в блокчейне находится на высоте 7, то есть блокчейн включает 8 блоков. Каждые 10 минут Лиза помещает последние неподтвержденные транзакции в новый блок и делает его доступным для всех, кому это интересно.

ДЛИНА БЛОКЧЕЙНА

Блокчейн в Биткоин содержит сотни тысяч блоков. На момент написания этих строк конечный блок имел высоту 550 836.

Блокчейн хранит транзакции точно так же, как электронная таблица. Но при этом каждый блок имеет *заголовок* для защиты целостности содержащихся в нем транзакций и предшествующих ему блоков. Предположим, что блокчейн на рис. 6.3 вырос и теперь содержит 20 блоков, поэтому конечный блок находится на высоте 19. На рис. 6.4 показано, что содержат последние несколько блоков в блокчейне.

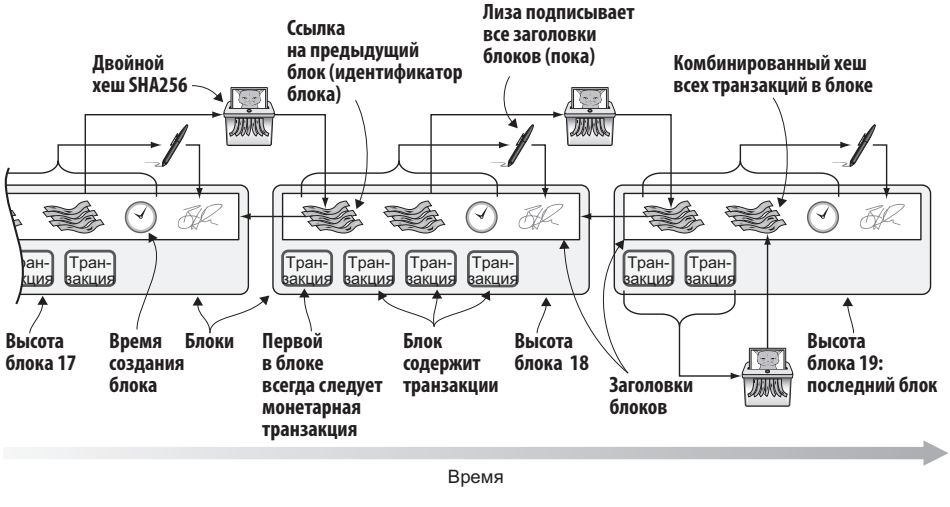
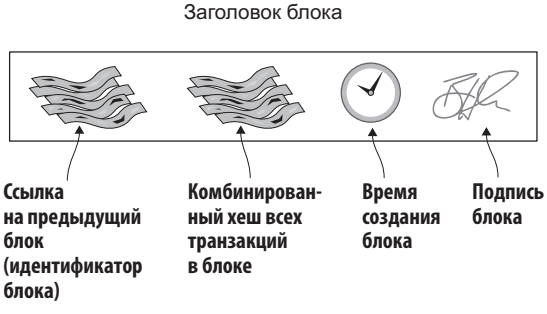


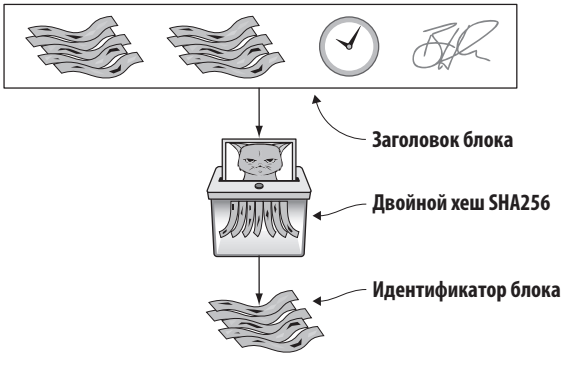
Рис. 6.4. Заголовок каждого блока защищает целостность транзакций в нем и во всех предшествующих блоках

Каждый блок содержит одну или несколько транзакций и заголовок. Заголовок блока включает:

- двойной хеш SHA256 заголовка предыдущего блока;
- комбинированный хеш транзакций в блоке, *корень дерева Меркла* (merkle root);
- время создания блока;
- подпись Лизы для заголовка блока.



Хеш заголовка блока служит идентификатором блока, так же как хеш транзакции служит идентификатором транзакции (txid). Время от времени я буду называть хеш заголовка блока *идентификатором блока*.



Первый левый элемент заголовка блока — это идентификатор предыдущего блока в блокчейне. Вот почему эта структура называется *цепочкой* блоков (блокчейном). Хеши предыдущих заголовков блоков образуют цепочку.

Второй элемент слева — это комбинированный хеш транзакций. Это *корень дерева Меркла*. Мы поговорим об этом элементе в последующих разделах в этой главе, а пока просто отмечу, что все транзакции в блоке хешируются в одно хеш-значение, которое записывается в заголовок блока. Нельзя изменить какую-либо транзакцию в блоке, не изменив корень дерева Меркла.

Третий элемент слева — время создания блока. Это время не является точным и даже не всегда увеличивается от блока к блоку. Но оно достаточно точное в человеческом понимании.

Четвертый элемент — подпись блока, оставленная Лизой, являющаяся своеобразной печатью «одобрено», которую может проверить каждый. Подпись Лизы доказывает, что она одобрила блок, и ее можно использовать как доказательство против Лизы, если та попытается смошенничать. Чуть ниже вы увидите, как это работает. Цифровая подпись в заголовке блока создает некоторые проблемы, которые мы исправим в главе 7, заменив цифровые подписи так называемым *доказательством работы*.

Лиза строит блок

Примерно каждые 10 минут Лиза создает новый блок, содержащий неподтвержденные транзакции. Она записывает этот блок в новый файл в общей папке. Право создавать новые файлы в общей папке имеют все, но никто не имеет права удалять или изменять файлы. Когда Лиза записывает блок в файл в общей папке, она *подтверждает* транзакции в этом блоке.

Допустим, Лиза собирается создать новый блок на высоте 20. Она должна:

1. Создать шаблон блока.
2. Подписать его.
3. Опубликовать.

ОБЩАЯ ПАПКА? СЕРЬЕЗНО?

Конечно же, в Биткоине нет никаких общих папок. В данном случае общая папка заменяет одноранговую сеть Биткоин, которую мы рассмотрим в главе 8.

Шаблоны блоков

Сначала Лиза создает *шаблон блока* — блок без подписи (рис. 6.5).

Она собирает несколько транзакций для включения в блок. Затем создает заголовок блока. Хеширует заголовок предыдущего блока, создавая его идентификатор, и включает результат в заголовок нового блока. Корень Меркла создается на основе транзакций, вошедших в шаблон блока, а в качестве времени создания выбирается текущее время.

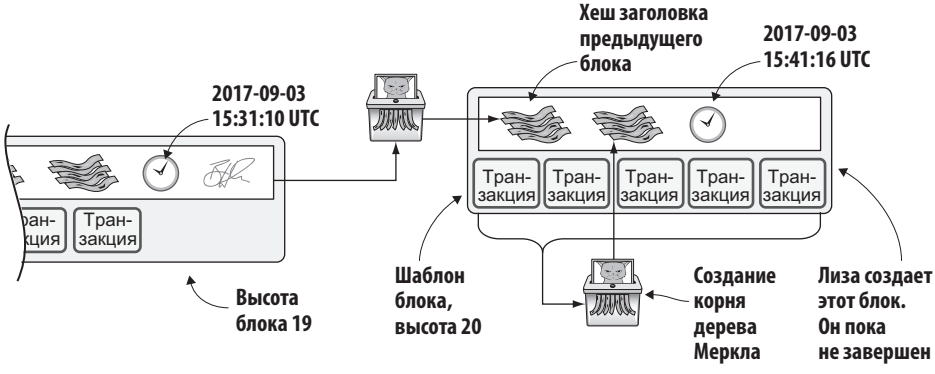


Рис. 6.5. Лиза создает новый блок. Он называется шаблоном блока, потому что пока не подписан

Первая транзакция в новом блоке — монетарная транзакция. Монетарные транзакции создают 50 CT в каждом блоке вместо 7200 CT, как это было в главе 5. Объясняется это просто: Лиза создает новый блок каждые 10 минут, то есть в течение 24 часов она создает 144 блока, что в сумме дает 7200 CT: 144 × 50 CT = 7200 CT. Подробнее о вознаграждении за блок и о монетарных транзакциях мы поговорим в главе 7.

ВОЗНАГРАЖДЕНИЕ ЗА БЛОК

В системе Биткоин к вознаграждению относятся не только вновь созданные монеты, но также комиссионные за транзакции, которые обсуждаются в главе 7. Вновь созданные монеты в блоке называются *блочной субсидией*.

Подписание блока

Прежде чем закончить создание блока, Лиза должна подписать его, используя закрытый ключ, который знает только она, как показано на рис. 6.6.

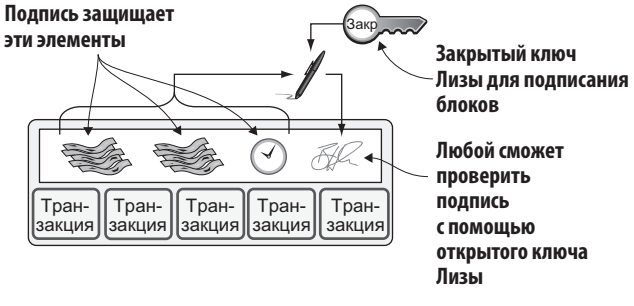


Рис. 6.6. Лиза подписывает блок своим закрытым ключом, предназначенным специально для подписания блоков. Парный ему открытый ключ известен всем сотрудникам

Используя свой закрытый ключ для подписания блоков, Лиза подписывает заголовок нового блока. Эта цифровая подпись утверждает:

- идентификатор предыдущего блока, то есть всю цепочку блоков перед этим новым блоком;
- корень дерева Меркла, то есть все транзакции в этом новом блоке;
- время создания блока.

Если что-то изменится в цепочке блоков перед новым блоком или в транзакциях в этом блоке, заголовок блока тоже должен будет измениться; следовательно, подпись станет недействительной.

ДОКАЗАТЕЛЬСТВО РАБОТЫ

Блоки в Биткоин подписываются иначе. Они «подписываются» с доказательством работы, как рассказывается в главе 7.

Открытый ключ, парный закрытому ключу для подписания блоков, должен быть доступен всем проверяющим. Компания может опубликовать открытый ключ в своей внутренней сети и на доске объявлений у главного входа. Подпись необходима, потому что только Лиза (пока) должна иметь возможность добавлять блоки в цепочку блоков. Например, Джон может создать блок и записать его в общую папку. Но он не сможет подписать его правильно, потому что у него нет закрытого ключа Лизы, поэтому никто не примет блок Джона.

Открытый ключ Лизы

```
020fe0e27cb1
d913c71970fc
07d1ce541c25
35e0d2d46828
9e3ec2ac66c8
78fd75
```

Использование закрытого ключа для подписания блоков — не лучшее решение, и вот почему:

- Закрытый ключ Лизы может быть украден. Если это случится, вор сможет создавать действительные блоки и записывать их в общую папку. Разумеется, монетарные транзакции в этих блоках будут переводить вознаграждение на РКН вора, а не Лизы.
- Источники, содержащие открытый ключ Лизы — например, доска объявлений и внутренняя сеть, — могут быть взломаны, и опубликованный открытый ключ может быть заменен открытым ключом злоумышленника. В этом случае некоторые проверяющие окажутся обманутыми и будут принимать блоки, подписанные ключом, отличным от ключа Лизы, предназначенного для подписания блоков. Таким незамысловатым способом злоумышленник сможет обмануть некоторую часть проверяющих. Сотрудники не должны доверять только листу бумаги на доске объявлений, потому что его легко

подменить листом с ложным открытым ключом. Сотрудники должны получить открытый ключ из разных источников, таких как доска объявлений, внутренняя сеть, и от других сотрудников. Злоумышленники слишком легко манипулируют единственным источником.

В главе 7 мы изменим способ подписания блоков, заменив цифровую подпись доказательством работы.

Публикация блока

После подписания Лиза должна опубликовать блок, сделав его доступным для проверяющих. Для этого она создает в общей папке новый файл с именем block_20.dat, в который записывает новый блок (рис. 6.7).

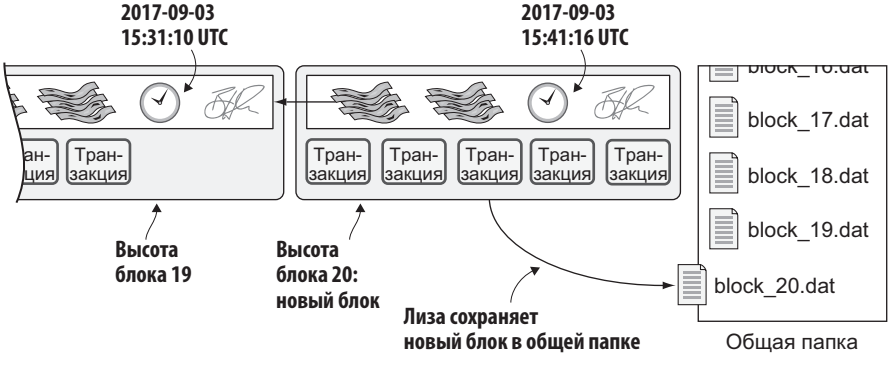


Рис. 6.7. Лиза подписывает новый блок и сохраняет его в новом файле в общей папке

Теперь блок опубликован. Любой желающий может прочитать этот блок из общей папки. Я уже говорил, что никто не может удалить или изменить этот файл из-за ограничения прав доступа к общей папке. Даже Лиза не сможет изменить его. Однако есть системный администратор, имеющий полные права на доступ к общей папке. Мы избавимся от системного администратора в главе 8, когда я представлю одноранговую сеть.

Выбор транзакций

Когда Лиза строит новый блок, то выбирает транзакции для включения в него. Она может выбрать любое число транзакций, от нуля до всех неподтвержденных транзакций. Порядок транзакций не важен, главное — чтобы все расходуемые транзакциями выходы присутствовали в блокчейне или в строящемся блоке. Например, блок на рис. 6.8 — вполне допустимый блок.

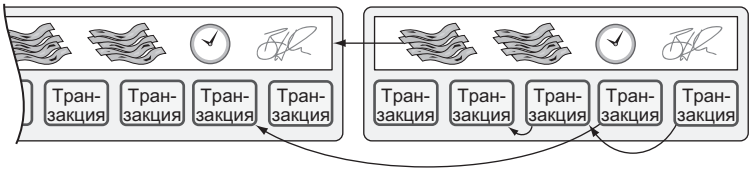


Рис. 6.8. Транзакции должны добавляться только в порядке расходования выходов. Никаких других ограничений не накладывается

Все транзакции в этом блоке расходуют выходы транзакций, уже включенных в блокчейн, то есть все они ссылаются на транзакции слева от себя. Но блок на рис. 6.9 является недействительным.

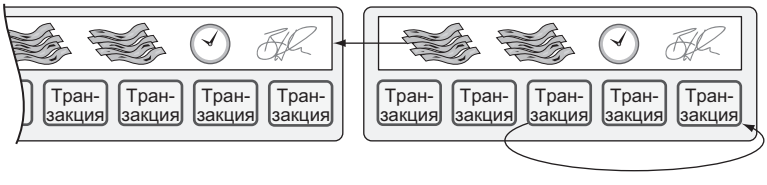


Рис. 6.9. Этот блок недействителен, потому что транзакция расходует выход, еще не включенный в блокчейн

Этот блок недействителен, потому что транзакция расходует выход, который находится после — справа от расходующей транзакции.

Как этот процесс защищает от удаления транзакций?

Допустим, Лиза решила съесть булочку, не заплатив за нее. Она создает транзакцию и помещает ее в текущий создаваемый блок, имеющий высоту 21. Она создает заголовок блока, подписывает его и записывает блок в новый файл (block_21.dat) в общей папке (рис. 6.10).

В кафе наблюдают за общей папкой, ожидая появления новых блоков. Когда Лиза сохраняет файл с блоком в общей папке, в кафе загружают его и проверяют. В том числе проверяются:

- действительность подписи заголовка блока с использованием открытого ключа Лизы, полученного на доске объявлений или во внутренней сети;
- существование идентификатора предыдущего блока, в данном случае блока 20;

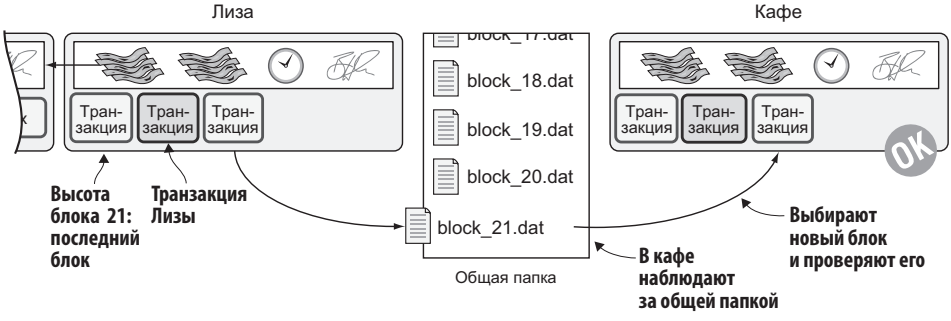


Рис. 6.10. Лиза создает блок, включающий ее оплату за булочку

- действительность всех транзакций в блоке с использованием того же подхода, что и в главе 5, когда проверяется множество неизрасходованных выходов транзакций (Unspent Transaction Outputs, UTXO);
- совпадение комбинированного хеша всех транзакций с корнем дерева Меркла в заголовке блока;
- нахождение времени создания блока в разумных пределах.

Лиза заплатила за булочку, а в кафе загрузили блок с транзакцией Лизы и проверили его. Кафе доставляет булочку Лизе, и она съедает ее.

Может ли Лиза отменить этот платеж, не опасаясь, что факт мошенничества будет доказан? Единственное, что она может сделать — создать другую, измененную версию блока 21, без своей транзакции и записать этот новый блок в общую папку как *block_21b.dat* (рис. 6.11).

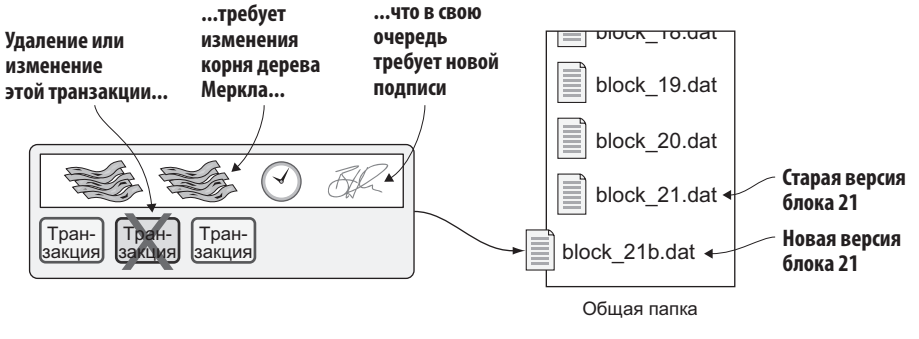


Рис. 6.11. Лиза создает альтернативный блок с высотой 21, не содержащий ее транзакции

Новая версия похожа на старую, но не содержит транзакции Лизы. Поскольку она вмешивается в транзакции в блоке, она должна заменить корень дерева Меркла в заголовке новым значением, соответствующим новому набору транзакций в блоке. Когда она меняет заголовок, подпись становится недействительной, поэтому заголовок требуется подписать заново. Чтобы сделать измененный блок доступным для проверяющих, Лиза должна поместить блок в общую папку, например в файле с именем block_21b.dat.

В кафе уже загрузили первую версию блока 21. Когда Лиза добавит новый файл с блоком, в кафе обнаружат, что в общей папке появилась другая версия блока (рис. 6.12).

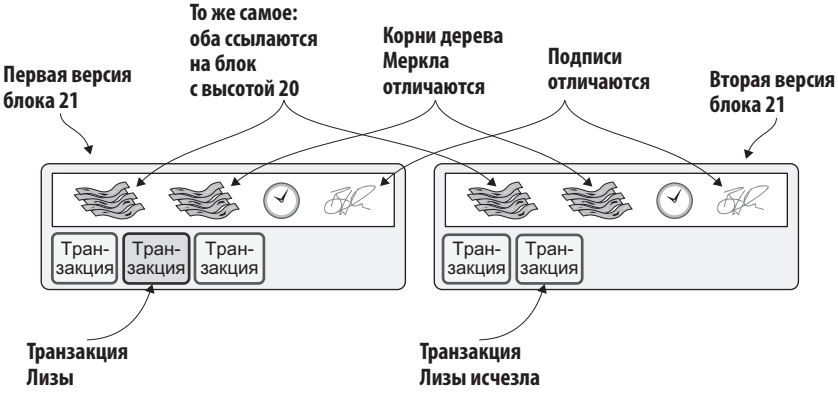


Рис. 6.12. В кафе увидят две версии блока 21, один с транзакцией Лизы и другой без нее

Теперь в кафе видят два разных блока на высоте 21, один из которых содержит перевод 10 CT в кафе, а другой — нет. Оба блока одинаково действительны, и ни один не является более точным с точки зрения проверки. Но теперь кафе может доказать, что Лиза играет в грязные игры, потому что она создала две разные *подписанные* версии блока. Подписи доказывают, что Лиза пошла на обман и это уже не та ситуация «слово против слова». Лиза будет уволена или, по крайней мере, отстранена от своей важной должности обработчика транзакций.

А что, если после 21 блока имелись другие блоки, когда Лиза пустилась на обман? Предположим, что в момент, когда Лиза решила удалить свою транзакцию, уже были созданы блоки 22 и 23 (рис. 6.13).

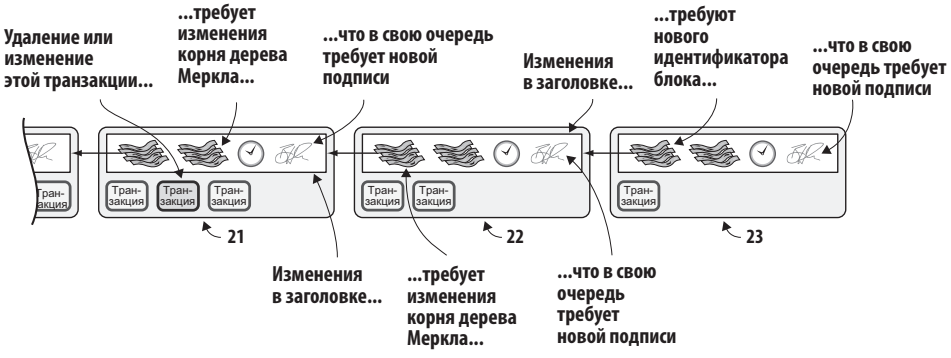
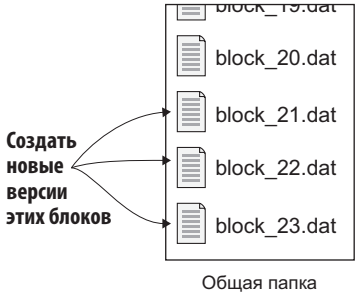


Рис. 6.13. Лизе нужно создать альтернативные версии блока с ее транзакцией, а также всех последующих блоков

Теперь ей нужно создать три альтернативных блока: 21, 22 и 23. Все они должны быть заменены действительными блоками.



Изменение чего-либо в блоке делает этот и все последующие блоки недействительными. Это связано с тем, что заголовок каждого блока содержит ссылку на предыдущий блок — идентификатор предыдущего блока, который станет недействительным при изменении предыдущего блока.

Какие преимущества дает блокчейн?

Блокчейн — это довольно сложный способ подписать пакет транзакций. Разве не проще, если Лиза будет каждые 10 минут подписывать сразу все транзакции, собранные в один большой блок? Это позволит достичь той же цели. Но этот подход имеет несколько проблем:

- по мере увеличения числа транзакций время, необходимое для подписания всего набора, будет увеличиваться;
- то же относится к проверяющим — время, необходимое для проверки подписи, увеличивается с количеством транзакций;
- проверяющим трудно узнать, что нового появилось со времени последней подписи. Эта информация ценна для поддержки множества UTXO.

При использовании блокчейна Лиза должна подписывать только самый последний блок транзакций и, косвенно через ссылку (идентификатор) на предыдущий блок, — все прошлые транзакции, как показано на рис. 6.14.

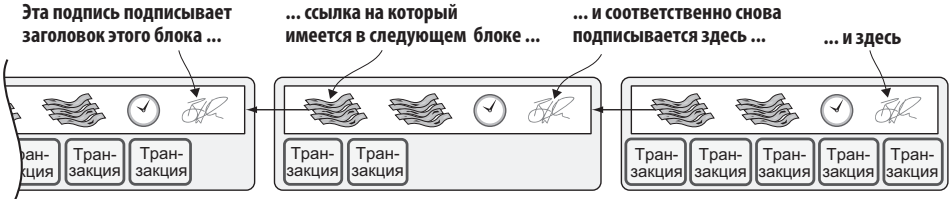


Рис. 6.14. Каждый блок одновременно подписывает все транзакции, созданные прежде, благодаря полю в заголовке с идентификатором предыдущего блока

Подпись в каждом блоке подкрепляет подписи в предыдущих блоках. Это станет особенно важным, когда в следующей главе мы заменим подписи доказательством работы.

Проверяющие также смогут легко увидеть, что нового появилось с момента публикации предыдущего блока, и соответствующим образом обновить свои множества UTXO. В новом блоке находятся только новые транзакции.

Блокчейн также предлагает несколько приятных дополнительных функций, о которых мы поговорим позже, таких как дерево Меркла.

Легкие кошельки

Сотрудники, желающие проверить блокчейн, чтобы убедиться в наличии действительной финансовой информации, используют программное обеспечение, которое загружает весь блокчейн и поддерживает множество UTXO в актуальном состоянии. Это программное обеспечение должно работать почти все время, чтобы своевременно получать новые блоки. Мы называем это программное обеспечение *полным узлом*. Полный узел знает обо всех транзакциях, начиная с блока 0, *первичного блока* (genesis block). Компания и кафе являются типичными полными узлами. Им не нужно доверять кому-то еще предоставление финансовой информации: они получают ее непосредственно из блокчейна. Любой желающий может свободно запустить это программное обеспечение у себя.

В главе 4 я представил мобильное приложение, которое сотрудники могут использовать для управления своими закрытыми ключами, а также для

отправки и получения денег. Это приложение-кошелек теперь адаптировано для поддержки новой системы блокчейн.

Большинство пользователей кошельков используют мобильные устройства с оплачиваемым трафиком, поэтому для них было бы нежелательно расходовать трафик на загрузку всех блоков, не представляющих интереса. Подавляющее большинство блоков не будет содержать транзакций, имеющих отношение к конкретным пользователям, поэтому загрузка всех данных приведет к тому, что их телефоны будут расходовать трафик без всякой пользы для них.

АЛЬТЕРНАТИВНЫЕ НАЗВАНИЯ

Прежде легкий кошелек назывался *SPV-клиентом*, или *SPV-кошельком*. SPV означает *Simplified Payment Verification — упрощенная проверка платежей*.

Разработчики полных узлов и кошельков тесно сотрудничают друг с другом, чтобы позволить кошелькам подключаться к полным узлам в Интернете, получать релевантные блоки и не расходовать огромные объемы трафика. Кошельки могут подключаться к любым полным узлам и запрашивать необходимые данные.

Предположим, что кошелек Джона содержит два адреса, $@_a$ и $@_b$, и он хочет получать уведомления от полного узла о транзакциях, касающихся его кошелька. Он может установить сетевое соединение с любым из полных узлов, например с кафе. Затем кошелек и полный узел начинают обмениваться информацией, как показано на рис. 6.15.

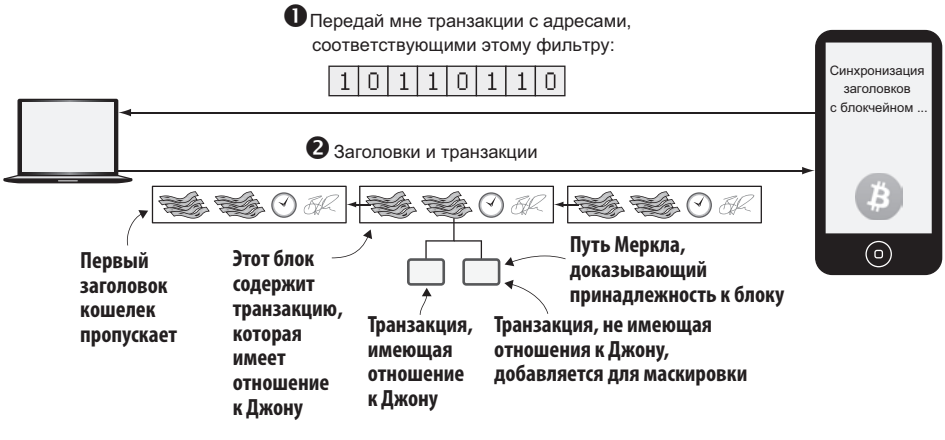


Рис. 6.15. Обмен информацией между легким кошельком и полным узлом. Полный узел посылает в кошелек заголовки всех блоков и часть транзакций

Подробнее о том, как устанавливается соединение и как кошелек и узел пересылают информацию друг другу, мы поговорим в главе 8. Здесь я лишь кратко опишу следующие основные моменты:

❶ Кошелек Джона запрашивает у полного узла все заголовки блоков, полученные после последнего заголовка известного кошельку, и все транзакции, имеющие отношение к адресам Джона.

❷ Полный узел, действующий в кафе, отправляет все запрошенные заголовки блоков в кошелек и все транзакции, имеющие отношение к адресам Джона.

BIP37

Этот процесс подробно описывается в BIP37. Описание можно найти на веб-ресурсе 9 (приложение B).

Выполняя шаг ❶, кошелек не посылает точного списка адресов Джона. Это навредило бы конфиденциальности Джона, потому что тогда в кафе узнали бы все адреса, принадлежащие Джону, и могли бы продать эту информацию страховой компании Acme Insurances. Поэтому кошелек Джона отправляет полному узлу фильтр. Этот фильтр называется *фильтром Блума* (bloom filter). Используя его, полный узел определяет, какие транзакции следует отправить в кошелек. Согласно фильтру, полный узел посылает все транзакции, относящиеся к адресам $@_a$ и $@_b$, а также транзакции, не имеющие отношения к кошельку Джона, благодаря чему маскируются адреса, на самом деле принадлежащие кошельку. Фильтры Блума не имеют ничего общего с блокчейном, но я все равно выделил для них отдельный подраздел, потому что они широко используются легкими кошельками.

На шаге ❷ в кошелек Джона отправляются транзакции и заголовки блоков, но полные блоки не отправляются (для экономии сетевого трафика). Однако одних только транзакций и заголовков блоков недостаточно, чтобы кошелек Джона мог убедиться, что транзакция действительно присутствует в блоке. Требуется нечто большее: *частичное дерево Меркла*, которое доказывает, что одна или несколько транзакций действительно включены в блок.

Эти два шага выполняются как этап синхронизации сразу после подключения кошелька к полному узлу в кафе. После этого, по мере создания новых блоков Лизой, полный узел в кафе будет забирать их и отправлять соответствующие заголовки в кошелек вместе со всеми транзакциями, касающимися адресов Джона, как было описано выше.

Далее мы обсудим фильтры Блума. Деревья Меркла будут описаны ниже, в разделе «Деревья Меркла».

Фильтры Блума маскируют адреса

Кошелек Джона содержит два адреса, $@_a$ и $@_b$, но Джон не хочет, чтобы кто-то узнал, что $@_a$ и $@_b$ принадлежат одному и тому же кошельку. У него есть основания осторожничать, потому что он слышал, что Acme Insurances платит хорошие деньги за такую информацию, позволяющую им «скорректировать» страховые выплаты, мотивируя это пристрастием страхуемого к употреблению выпечки.

Создание фильтра Блума

Чтобы усложнить определение принадлежности адресов, кошелек Джона создает фильтр Блума для отправки на полный узел (рис. 6.16).

Рис. 6.16. Клиент посылает полному узлу фильтр Блума, чтобы замаскировать адреса, принадлежащие кошельку

Фильтр Блума — это последовательность битов, которые, как упоминалось в главе 2, могут иметь значение 0 или 1. Длина фильтра Блума составляет 8 бит. На рис. 6.17 показано, как он был создан.

Кошелек создает последовательность битов (фильтр Блума) и инициализирует их нулями. Затем добавляет в фильтр хеши всех открытых ключей (PKH) Джона, начиная с PKH_a — PKH для $@_a$.

Он пропускает PKH_a через первую из трех хеш-функций, которая возвращает число 2. Затем это

ПОЧЕМУ ТРИ ХЕШ-ФУНКЦИИ?

Количество хеш-функций может быть любым, как и размер фильтра Блума. В этом примере используются три хеш-функции и фильтр длиной 8 бит.

число интерпретируется как индекс бита в фильтре Блума, и бит с индексом 2 (третий слева) устанавливается в значение `1`. Затем PKH_a пропускается через вторую хеш-функцию, которая возвращает `0`, и соответствующий бит (первый слева на рис. 6.17) устанавливается в значение 1. Наконец, третья хеш-функция возвращает `6`, и бит с индексом 6 (седьмой слева) устанавливается в 1.

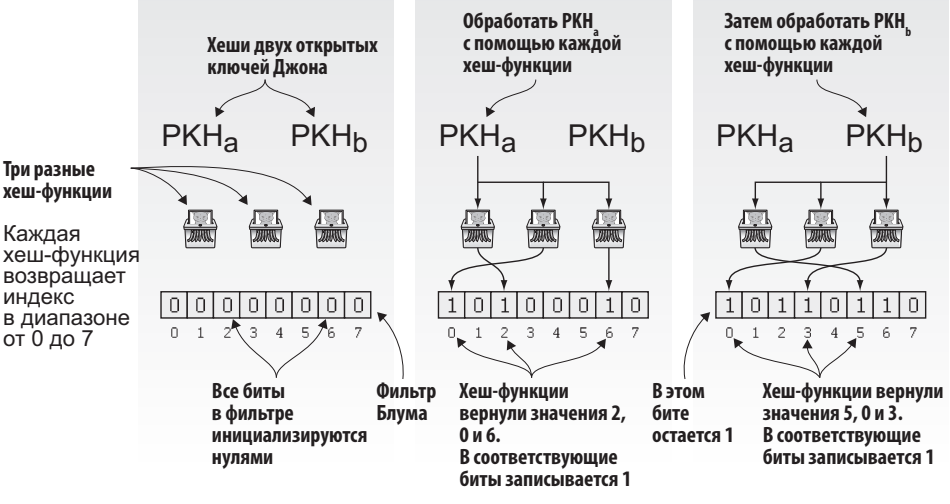


Рис. 6.17. Легкий кошелек создает фильтр Блума для отправки полному узлу. В фильтр Блума добавляются все адреса, имеющиеся в кошельке

Далее приходит черед хеша PKH_b, который обрабатывается точно так же. Три хеш-функции возвращают `5`, `0` и `3`. Все эти три бита устанавливаются в значение `1`. Обратите внимание, что бит с индексом 0 уже был установлен в ходе обработки PKH_a, поэтому его значение не изменяется.

На этом создание фильтра Блума завершается, и он отправляется полному узлу.

Использование фильтра Блума

Полный узел получает фильтр Блума от кошелька и использует его для фильтрации транзакций, отправляемых в кошелек.

Предположим, что Лиза только что опубликовала новый блок в общей папке и полный узел проверил его. Теперь полный узел должен отправить в кошелек заголовок нового блока и все соответствующие транзакции в нем.

Как полный узел использует фильтр Блума, чтобы выбрать транзакции для отправки?

Блок содержит три транзакции: Tx_1, Tx_2 и Tx_3 (рис. 6.18).

Рис. 6.18. Блок, выбранный для отправки, содержит три транзакции, но только одна имеет отношение к Джону

Транзакции Tx_1 и Tx_3 не имеют ничего общего с адресами Джона, но Tx_2 — это платеж на адрес Джона $@_b$. Давайте посмотрим, как полный узел использует фильтр Блума (рис. 6.19).

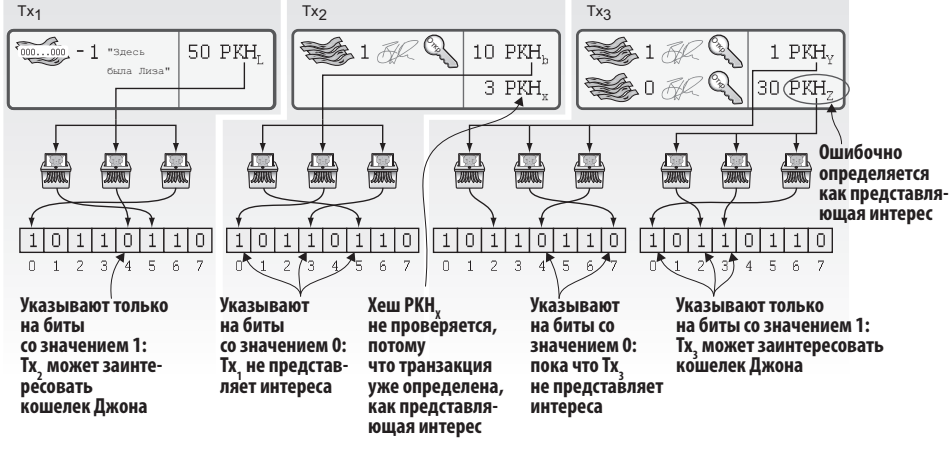


Рис. 6.19. Полный узел использует фильтр Блума для определения транзакций, которые «заинтересуют» кошелек

Полный узел проверяет каждый выход в транзакции, сопоставляя PKH с фильтром. Он начинает с транзакции Tx_1, которая имеет один выход с адресом PKH_L. Чтобы проверить соответствие PKH_L фильтру, он пропускает PKH_L через те же три хеш-функции, которые использовались кошельком Джона для создания фильтра. Хеш-функции возвращают индексы 5, 1

и 0. Биты с индексами 5 и 0 имеют значение 1, но бит с индексом 1 имеет значение 0. Значение 0 указывает, что $РКН_L$ определенно не представляет интереса для кошелька Джона. Если бы кошелек Джона интересовался адресом $РКН_L$, он добавил бы его в фильтр, установив бит 1 в значение 1. Поскольку $РКН_L$ — единственный РКН в $Тх_1$, эта транзакция неинтересна кошельку Джона.

Далее проверяется транзакция $Тх_2$. Она содержит два РКН: $РКН_b$ и $РКН_X$. Первым проверяется $РКН_b$. Хеш-функции возвращают для этого РКН индексы 5, 0 и 3. Все три бита с этими индексами имеют значение 1. В этом случае узел не может с уверенностью утверждать, что эта транзакция будет интересна кошельку, но он не может утверждать, что она *точно не* интересна. Проверять другие РКН в этой транзакции не имеет смысла, поскольку узел уже определил, что транзакция $Тх_2$ должна быть отправлена в кошелек.

Последняя транзакция имеет два выхода: с адресами $РКН_Y$ и $РКН_Z$. Первым проверяется $РКН_Y$, который указывает на биты с индексами 2, 7 и 4. Биты 4 и 7 имеют значение 0, а значит, $РКН_Y$ определенно не интересен кошельку. Далее проверяется $РКН_Z$, для которого хеш-функции возвращают 2, 3 и 0. Все три бита с этими индексами в фильтре имеют значение 1. Это опять же означает, что транзакция $Тх_3$ *может быть* интересна кошельку, поэтому узел отправит и эту транзакцию. На самом деле кошелек Джона не содержит $РКН_Z$, но фильтр Блума стремится соответствовать более широкому кругу адресов, чем нужно, чтобы обеспечить некоторую степень конфиденциальности. Мы называем это *ложным срабатыванием*.

В результате фильтрации Блума узел отправит в кошелек транзакции $Тх_2$ и $Тх_3$. Как отправляются транзакции — это совершенно другая история, описанная в разделе «Деревья Меркла».

ВНИМАНИЕ

Далее следуют сложные рассуждения. Вы можете смело пропустить их и перейти к разделу «На чем мы остановились?».

Предыдущее описание дает лишь упрощенное представление о происходящем в действительности. Выше мы проверили только РКН, указанные в выходах транзакций. То есть таким способом будут выявлены транзакции, соответствующие платежам по любому из адресов Джона. А как быть с транзакциями, которые *расходуют* средства с адресов Джона? Можно было бы подумать, что полный узел не должен отправлять эти транзакции в кошелек,

потому что тот уже знает о них, так как сам создал их. К сожалению, эти транзакции должны отправляться по двум важным причинам.

Во-первых, транзакции могли быть созданы не этим приложением кошелька. Джон может пользоваться несколькими кошельками, генерирующими адреса из одного начального числа (семени). Например, вспомните, как в главе 4 рассказывалось, что кошелек можно восстановить из мнемонического предложения. Это предложение можно использовать в нескольких приложениях кошелька одновременно. Джон может выполнить платеж в одном из приложений и получить уведомление о платеже в другом, что позволит ему следить за общим балансом в этом приложении.

Во-вторых, Джон должен получить уведомление, когда транзакция будет подтверждена. Транзакция уже может иметься в приложении кошелька, но она по-прежнему будет помечена как *неподтвержденная*. Джон должен знать, когда транзакция будет включена в блок, поэтому ему важно, чтобы узел отправил ему эту транзакцию, когда она окажется в блоке.

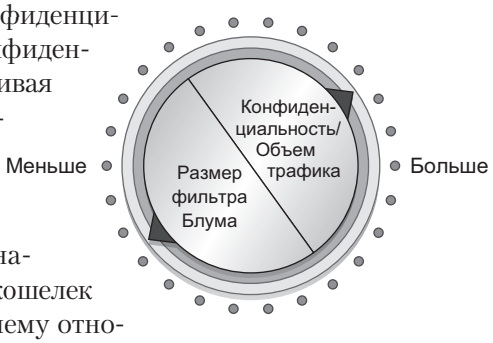
Вот что на самом деле проверяет узел (рис. 6.20):

- идентификатор (txid) транзакции;
- все выходы транзакций (TXO), ссылки на которые присутствуют во входах;
- все элементы данных в сценариях подписи;
- все элементы данных в выходах.

Рис. 6.20. Чтобы определить заинтересованность в транзакции, с помощью фильтра Блума проверяется несколько компонентов

Чтобы получать уведомления о расходах, кошелек Джона должен добавить в фильтр Блума все свои открытые ключи или все ссылки UTXO.

Управление конфиденциальностью и объемом трафика



Цель фильтра Блума — повысить конфиденциальность пользователей. Уровнем конфиденциальности можно управлять, настраивая соотношение между количеством единиц в фильтре и размером фильтра. Чем больше отношение количества единиц в фильтре к его размеру, тем больше ложных срабатываний, а значит, полный узел будет отправлять в кошелек больше транзакций, не имеющих к нему отношения. Чем больше транзакций отправляется, тем больший объем трафика будет потрачен впустую и тем выше уровень конфиденциальности.

Давайте выполним некоторые несложные вычисления для иллюстрации. Фильтр Блума в предыдущем примере имеет 8 бит, пять из которых равны `1`. Вероятность, что каждая хеш-функция вернет индекс, соответствующий биту со значением 1, равна 5/8. Отсюда вероятность, что все три хеш-функции в одной проверке вернут индексы битов со значением 1, равна $(5/8)^3$. Вероятность, что проверка даст отрицательный результат — по крайней мере, одна из трех хеш-функций вернет индекс бита со значением 0, — составляет $1 - (5/8)^3$. Полный узел будет выполнять несколько проверок для каждой транзакции, обычно девять для транзакции с двумя входами и двумя выходами. Проверим это по списку элементов, проверяемых полным узлом:

- идентификатор (txid) транзакции (1);
- все ссылки ТХО во входах (2);
- все элементы данных в сценариях подписи (открытый ключ и подпись × × 2 = 4);
- все элементы данных в выходах (2).

Вероятность того, что все девять проверок дадут отрицательный результат, составляет $(1 - (5/8)^3)^9 \approx 0{,}08$. То есть в кошелек будут отправляться почти все — 92/100 — транзакции. Эти расчеты наглядно показывают, что присутствие всего трех нулей в 8-битном фильтре Блума не поможет значительно уменьшить объем передаваемых данных, но лучше защищает вашу конфиденциальность.

Чтобы получить меньше ложных срабатываний, кошелек Джона должен использовать фильтр Блума большего размера, чтобы уменьшить отношение (количество единиц/размер фильтра Блума).

Определим несколько обозначений:

t = количество проверок, выполняемых с одной транзакцией (9);

p = вероятность, что транзакция будет признана неинтересной;

r = отношение количества единиц к размеру фильтра Блума.

Теперь вычисления, произведенные выше, можно представить в виде следующей формулы:

$$(1-r^3)^t = p \Rightarrow 1-r^3 = p^{\frac{1}{t}} \Rightarrow r^3 = 1-p^{\frac{1}{t}} \Rightarrow r = \sqrt[3]{1-p^{\frac{1}{t}}}.$$

Допустим, вы хотите получить только 1/10 всех транзакций (при условии, что все транзакции похожи на предыдущую, то есть имеют по два входа и два выхода). Насколько большим должен быть фильтр Блума в этом случае?

$$t = 9,\ p = \frac{9}{10}$$
$$r = \sqrt[3]{1-p^{\frac{1}{t}}} = \sqrt[3]{1-(\frac{9}{10})^{\frac{1}{9}}} \approx 0,23.$$

Согласно полученному результату, фильтр Блума должен иметь длину около $6/0,23 \approx 26$ бит, чтобы получить только 1/10 всех транзакций. Длина фильтра должна быть кратна 8 битам, поэтому 26 бит следует округлить до 32.

Имейте в виду, что это довольно грубый расчет, основанный на нескольких, не всегда верных предположениях относительно характеристик транзакций. Мы также не учитываем, что число единиц в примере не всегда строго равно шести, а может изменяться в диапазоне от трех до шести, учитывая, что оба адреса Джона могут генерировать одинаковые наборы индексов. Тем не менее этот пример помогает получить представление о том, насколько большим должен быть фильтр Блума.

Проблемы с фильтром Блума

Фильтры Блума широко используются многими легкими кошельками, но они страдают некоторыми специфическими проблемами:

- *Конфиденциальность* — узел, который получает фильтры Блума от облегченного клиента, может с высокой точностью определить, какие адреса принадлежат кошельку. Чем больше фильтров, тем выше точность. Подробнее об этом рассказывается на веб-ресурсе 14 (приложение B).

- *Производительность* — впервые получив фильтр Блума от облегченного клиента, полный узел должен просканировать весь блокчейн в поисках подходящих транзакций. Для такого сканирования требуется много памяти и до нескольких минут времени, в зависимости от производительности аппаратного обеспечения узла. Этот факт можно использовать для атак типа *отказ в обслуживании* (Denial-of-Service, DoS) на полные узлы, чтобы заставить их перестать отвечать на запросы клиентов.

В предложениях по улучшению Биткоин (Bitcoin Improvement Proposal, BIP), BIP157 и BIP158, были представлены способы решения этих проблем, но они пока не получили широкого распространения. Основная идея заключается в том, чтобы использовать обратный процесс, когда полный узел посылает в облегченный кошелек фильтр для каждого блока. Этот фильтр описывает адреса, которые затрагивает блок. Облегченный клиент проверяет, соответствуют ли его адреса фильтру, и если соответствуют, загружает весь блок. Блок можно получить из любого источника, а не только из полного узла, который отправил фильтр.

На чем мы остановились?

Для ориентации на рис. 6.21 показана часть схемы из раздела «Шаг ❹: Кошельки» в главе 1, где кошелек Боба уведомляется о переводе денег от Алисы.

Рис. 6.21. Кошелек Биткоин уведомляется полным узлом о поступлении платежа

В примере в этой главе Джон отправлял фильтр Блума полному узлу в кафе, чтобы получать информацию, касающуюся только его. Полный узел получил блок, содержащий две транзакции, которые могут заинтересовать Джона, по крайней мере, если судить по фильтру, который указал Джон.

Затем заголовок нового блока и потенциально интересные транзакции посылаются в кошелек Джона.

Деревья Меркла

Теперь, определив транзакции для отправки в кошелек, полный узел должен послать заголовок нового блока и все транзакции, которые могут заинтересовать кошелек Джона.

Рис. 6.22. Полный узел пересылает в кошелек заголовок блока и потенциально релевантные транзакции

Полный узел определил, что в кошелек следует отправить транзакции Tx_2 и Tx_3. Однако по заголовку и двум транзакциям кошелек Джона не сможет проверить принадлежность транзакций блоку. Корень дерева Меркла зависит от трех транзакций — Tx_1, Tx_2 и Tx_3, — а кошелек получает от полного узла только Tx_2 и Tx_3. Кошелек не сможет воссоздать корень дерева Меркла в заголовке блока. Ему нужно больше информации, чтобы убедиться, что транзакции действительно включены в блок. Как вы помните, наша цель — сэкономить трафик, поэтому отправка всех транзакций в блоке — не лучшее решение.

Создание корня дерева Меркла

Пришло время показать, как Лиза создавала корень дерева Меркла. Предположим, Лиза создает заголовок блока, показанный на рис. 6.22. Она должна

вычислить комбинированный хеш всех транзакций, называемый корнем дерева Меркла (рис. 6.23). Чтобы вычислить корень, она создает иерархию криптографических хешей — дерево Меркла.

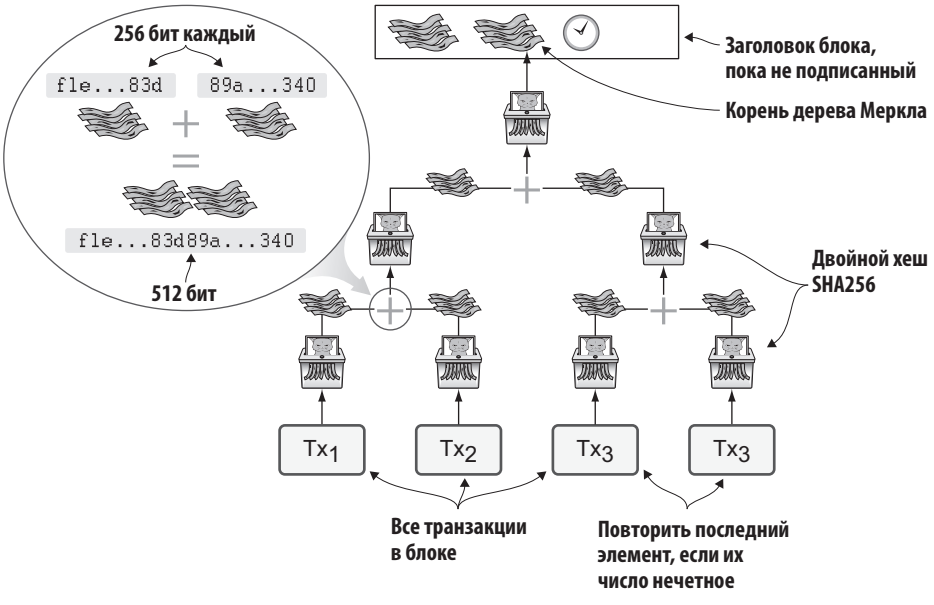


Рис. 6.23. Лиза вычисляет корень дерева Меркла из транзакций в блоке

Транзакции располагаются в том же порядке, как в блоке. Если количество элементов нечетное, последний из них повторяется и добавляется в конец. Этот дополнительный элемент не добавляется в блок — он используется только для вычисления дерева Меркла.

Каждый элемент (в данном случае каждая транзакция) дважды хешируется с применением функции SHA256. В результате получается четыре хеш-значения по 256 бит каждое.

Хеш-значения *объединяются* попарно, то есть для каждой пары второй хеш добавляется в конец первого. Например, в результате объединения `abc` и `def` получается `abcdef`.

Из четырех хеш-значений в результате получается два объединенных значения. Поскольку 2 — четное число, нет необходимости добавлять дополнительный элемент в конец. Каждое из двух объединенных значений снова хешируется, и в результате получаются два 256-битных хеша.

Эти два хеша объединяются в одно 512-битное значение. Это значение снова хешируется, и получается 256-битный корень дерева Меркла, который записывается в заголовок блока. Если какая-то транзакция добавится, удалится или изменится, корень дерева Меркла необходимо пересчитать (рис. 6.24).

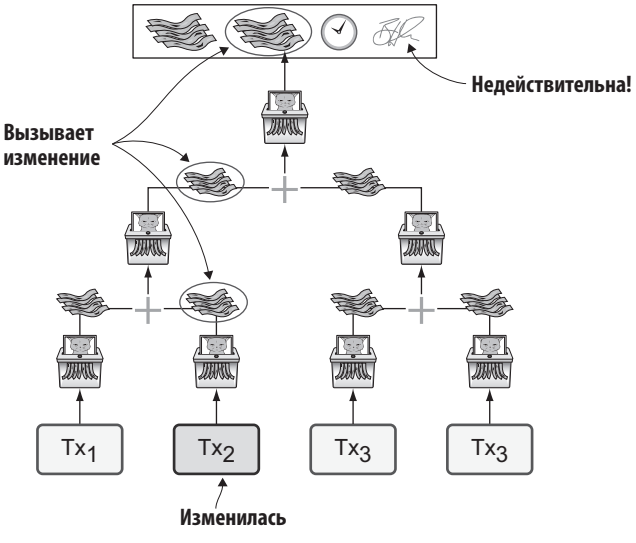


Рис. 6.24. Любые изменения в транзакциях вызывают изменение корня дерева Меркла, что делает подпись недействительной

Это хорошо, потому что если кто-то вмешается в транзакции после того, как Лиза подпишет заголовок блока, подпись станет недействительной.

Доказательство включения транзакции в блок

Полный узел должен послать транзакции Tx_2 и Tx_3 в кошелек Джона, потому что считает, что они представляют для него интерес. Кроме того, полный узел должен доказать, что обе транзакции, Tx_2 и Tx_3, включены в блок. Но для начала рассмотрим доказательство включения только одной транзакции, Tx_2. Более объемный и сложный пример мы рассмотрим позже в этой главе.

Как полный узел может доказать, что транзакция Tx_2 включена в блок? Он может передать *частичное дерево Меркла*, связывающее Tx_2 с корнем

Меркла в заголовке блока. Идея состоит в том, чтобы отправить в кошелек минимальный объем информации, с помощью которого можно убедиться, что Tx_2 включена в блок. В этом примере узел отправит в кошелек данные, перечисленные на рис. 6.25.

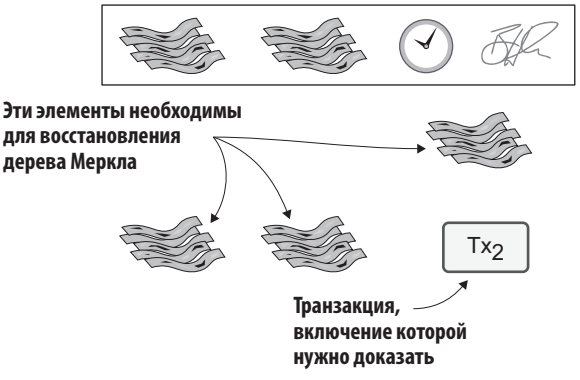


Рис. 6.25. Минимальный объем информации, доказывающий включение Tx_2 в блок. Эту информацию полный узел посылает в кошелек

Легкий кошелек использует эту информацию для проверки включения Tx_2 в блок, вычисляя промежуточные хеши в направлении к корню и убеждаясь, что хеш Tx_2 находится среди хешей, переданных полным узлом (рис. 6.26).

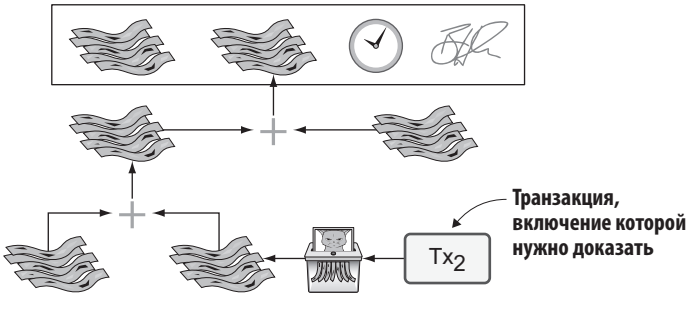


Рис. 6.26. Легкий кошелек проверяет включение Tx_2 в блок, реконструируя корень дерева Меркла

Я убрал хеш-функции из диаграммы, чтобы упростить ее. Теперь кошелек может быть «уверен», что Tx_2 находится в блоке.

Как выполняется проверка в действительности

ВНИМАНИЕ

Далее подробно описывается, как создать и проверить частичное дерево Меркла. Если вам это неинтересно, можете сразу перейти к разделу «Безопасность легких кошельков».

Создание частичного дерева Меркла

Частичное дерево Меркла — это урезанная версия полного дерева, содержащая только части, необходимые для доказательства включения Tx_2 в дерево. Полный узел отправляет в кошелек три элемента:

- заголовок блока;
- частичное дерево Меркла;
- транзакцию Tx_2.

Попробуем воссоздать частичное дерево Меркла. Полному узлу известно общее число транзакций в блоке, поэтому ему известна форма дерева Меркла. Чтобы построить частичное дерево Меркла, полный узел проверяет хеши в дереве, начиная с корня и перемещаясь вниз по дереву, сначала в левую ветвь (рис. 6.27).

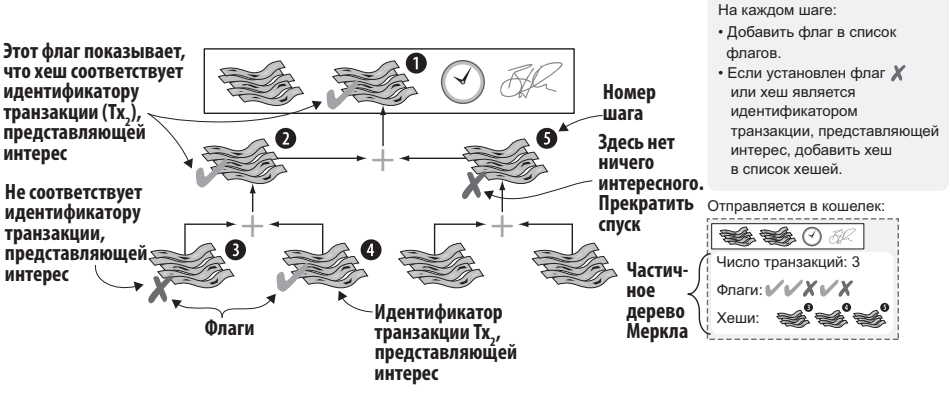


Рис. 6.27. Полный узел создает частичное дерево Меркла, связывающее Tx_2 с корнем дерева Меркла в заголовке блока

Частичное дерево Меркла включает:

- число, определяющее общее количество транзакций в блоке;
- список флагов;
- список хешей.

Как отмечено в следующей таблице, на каждом шаге с текущим хешем выполняются две операции.

1. Флаг добавляется в список. Значок × означает, что далее в этой ветви дерева нет ничего интересного; значок ✓ означает, что эта ветвь содержит транзакцию, представляющую интерес.
2. Если установлен флаг × или хеш является идентификатором транзакции, представляющей интерес, этот хеш добавляется в список хешей.

Шаг	Соответствует идентификатору транзакции, представляющей интерес?	Список флагов	Установлен флаг × или хеш является идентификатором транзакции, представляющей интерес?	Список хешей
❶	Да	✓	Нет	–
❷	Да	✓✓	Нет	–
❸	Нет	✓✓×	Да	3
❹	Да	✓✓×✓	Да	3 4
❺	Нет	✓✓×✓×	Да	3 4 5

Такой порядок шагов называется *обходом дерева в глубину*, то есть сначала выполняется движение вниз по дереву настолько, насколько возможно, а затем производится переход к следующей ветви. Но для нас нет смысла выполнять полный обход ветвей, в которых нет транзакций, представляющих интерес. Эти ветви отмечены в списке флагов как ×. Выполняя обход, полный узел останавливается, достигнув флага ×, потому что нет необходимости отправлять ненужные данные в кошелек, отсюда и термин «*частичное* дерево Меркла».

Создав частичное дерево Меркла, полный узел отправит его и заголовок блока в кошелек, а также фактическую транзакцию Tx_2. Заголовок блока с частичным деревом Меркла часто называют *доказательством Меркла*.

Проверка частичного дерева Меркла

Кошелек получает от полного узла заголовок блока, частичное дерево Меркла и транзакцию Tx_2. Это все, что ему нужно, чтобы убедиться, что Tx_2 действительно включена в блок. Далее кошелек должен проверить существование пути, «связывающего» Tx_2 с корнем дерева Меркла в заголовке блока. Сначала проверяется частичное дерево Меркла (рис. 6.28).

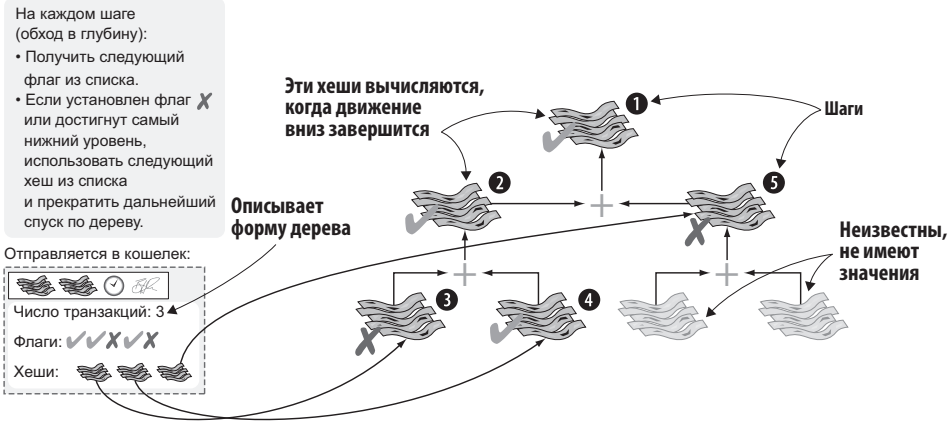


Рис. 6.28. Кошелек проверяет частичное дерево Меркла

Исходя из количества транзакций (три), полученного от полного узла, выстраивается структура дерева Меркла. Кошелек знает, как выглядит дерево Меркла с тремя транзакциями.

Списки флагов и хешей используются, чтобы прикрепить хеши к дереву Меркла в порядке обхода в глубину, как показано ниже.

Шаг	Следующий флаг в списке	Оставшиеся флаги в списке	Установлен флаг × или достигнут самый нижний уровень?	Прикрепить хеш	Список хешей
❶	✓	✓×✓×	Нет	–	3 4 5
❷	✓	×✓×	Нет	–	3 4 5
❸	×	✓×	Да	3	4 5
❹	✓	×	Да	4	5
❺	×		Да	5	

Теперь кошелек имеет достаточное количество хешей (❸, ❹ и ❺), прикрепленных к дереву Меркла, чтобы заполнить пробелы вверх, к корню частичного дерева Меркла. Сначала вычисляется хеш на шаге ❷ из ❸ и ❹; затем вычисляется корень из ❷ и ❺.

Потом вычисленный корень Меркла сравнивается с фактическим корнем Меркла в заголовке блока, чтобы убедиться, что они совпадают, и проверяется присутствие хеша Tx_2 в списке хешей, полученных от полного узла (рис. 6.29).

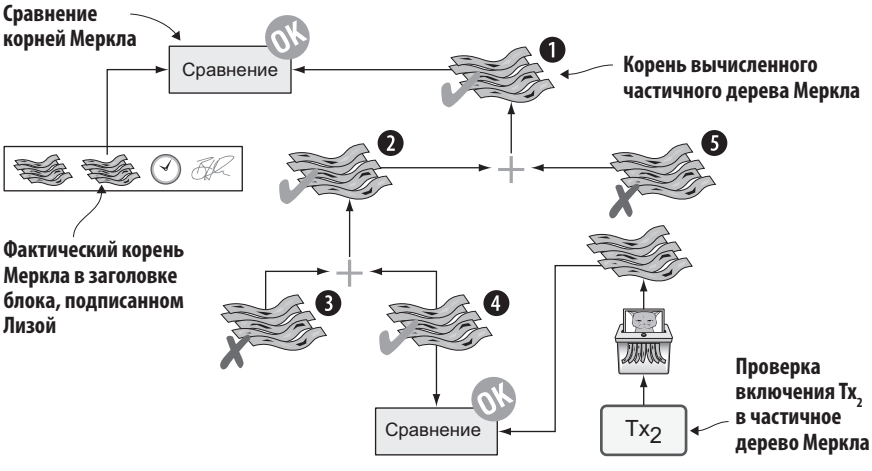


Рис. 6.29. Кошелек проверяет совпадение корней Меркла и присутствие Tx_2 в списке хешей. Успешность проверок доказывает, чтоTx_2 включена в блок

Если среди хешей в частичном дереве Меркла обнаружится совпадение с транзакцией и если корень частичного дерева Меркла совпадет с корнем Меркла в заголовке блока, можно считать, что полный узел доказал включение Tx_2 в блок.

Но полный узел должен отправить две транзакции из этого блока. Как будет выглядеть доказательство Меркла с двумя транзакциями? Ему придется послать несколько доказательств Меркла? Конечно нет, но мы обсудим этот случай в конце главы, в разделе с упражнениями.

Обработка тысяч транзакций в блоке

Блок в предыдущем примере содержал всего три транзакции. Мы не получили большой экономии, отправляя заголовок, частичное дерево Меркла

и Tx_2. С таким же успехом можно отправить идентификаторы всех трех транзакций вместо частичного дерева Меркла — это было бы намного проще. Тем не менее доказательство Меркла позволяет получить существенный выигрыш с увеличением числа транзакций в блоке.

Предположим, что полный узел только что проверил блок, содержащий 12 транзакций. Сопоставив транзакции с фильтром Блума кошелька, он установил, что две транзакции могут быть ему интересны. Этот случай показан на рис. 6.30.

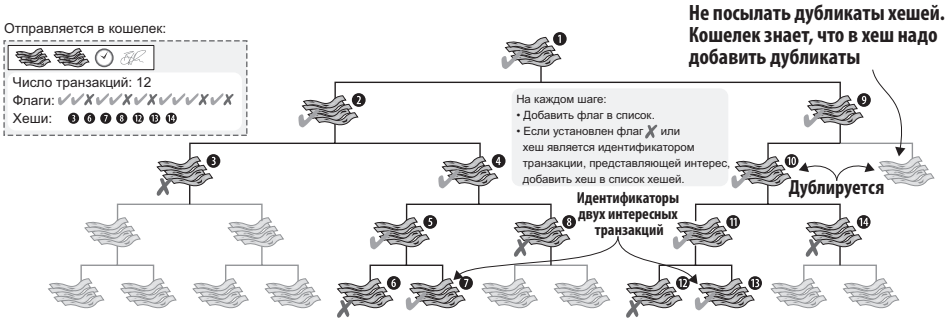


Рис. 6.30. Конструирование частичного дерева Меркла для 12 транзакций, две из которых представляют интерес

Полный узел должен отправить только заголовок блока, число 12, 14 флагов и 7 хешей. Это составит около 240 байт, что намного меньше, чем заголовок блока и все 12 идентификаторов транзакций (около 464 байт).

Давайте прикинем, какой выигрыш дает отправка доказательства Меркла по сравнению с упрощенным решением, когда отправляется полный блок и все идентификаторы транзакций, с увеличением количества транзакций (табл. 6.1).

В табл. 6.1 предполагается, что все транзакции имеют длину 250 байт, и требуется доказать включение только одной транзакции. Размер блока вычисляется как 80 байт заголовка блока плюс количество транзакций, умноженное на 250. Упрощенное доказательство вычисляется как 80 байт заголовка блока плюс количество транзакций, умноженное на 32. Размер доказательства Меркла вычисляется как 80 байт заголовка блока плюс длина списка хешей, умноженная на 32. Здесь не учитываются флаги и количество транзакций, потому что они дают незначительный прирост.

Таблица 6.1. Размеры доказательства Меркла и посылки в упрощенном решении для блоков с разным числом транзакций

Число транзакций в блоке	Размер блока (в байтах)	Размер посылки для упрощенного решения (в байтах)	Размер доказательства Меркла (в байтах)	Длина списка хешей
1	330	112	112	1
10	2580	400	240	5
100	25 080	3280	336	8
1000	250 080	32 080	432	11
10 000	2 500 080	320 080	560	15
100 000	25 000 080	3 200 080	656	18

80-БАЙТНЫЙ ЗАГОЛОВОК

В системе Биткоин заголовок блока всегда содержит 80 байт. Заголовки блоков в системе жетонов на булочки немного больше из-за подписи. В следующей главе мы изменим заголовок блока, чтобы приблизить его к системе Биткоин; а в главе 11 мы поговорим о версиях, которые также включаются в заголовки блоков.

Размер доказательства Меркла растет не так быстро, как размер посылки в случае упрощенного решения, потому что размер доказательства Меркла находится в *логарифмической* зависимости от количества транзакций, тогда как размер посылки в упрощенном решении — в *линейной*. Когда блок *удваивается* в размере, размер доказательства Меркла увеличивается на *постоянную величину*, примерно равную 32 байтам, тогда как размер посылки в упрощенном решении удваивается.

Безопасность легких кошельков

Легкие кошельки — отличная находка для системы жетонов на булочки. Но пользователи должны знать, чего им не хватает, по сравнению с полными узлами.

Полные узлы проверяют всю историю блокчейна и точно знают, что деньги, которые расходует транзакция, действительно существуют и подписи действительны.

Легкий кошелек знает всю цепочку заголовков блоков. Он проверит правильность подписания каждого заголовка Лизой. Когда кошелек получает транзакцию и доказательство Меркла, он может проверить присутствие транзакции в блоке и подпись Лизы. Но не может проверить многое другое. Например:

- что все программы на языке Script в транзакции возвращают признак «ОК», который обычно проверяет подписи во входах;
- что средства, расходуемые выходами, еще не израсходованы;
- что получены все релевантные транзакции.

Легкий кошелек также не знает, какими правилами руководствуется полный узел. Полный узел может использовать правило, которое выплачивает двойную награду Лизе. Типичный полный узел будет рассматривать любой блок, выплачивающий Лизе слишком много, как недействительный, потому что это противоречит его правилам, и будет отбрасывать блок.

Легкий кошелек должен доверять полному узлу проверку всего этого от его имени и верить, что полный узел следует тем ожидаемым правилам.

Полный узел может скрывать релевантные транзакции от кошелька и не уведомлять о некоторых входящих или исходящих транзакциях.

Легкий кошелек перекладывает ответственность за проверку на полный узел, к которому подключен. Предположим, что Лиза создала недопустимый блок, например блок, содержащий транзакцию, которая расходует несуществующий выход. Получив такой блок, полный узел должен проверить и удалить его, потому что он недействителен. Но может так случиться, что полный узел, преднамеренно или случайно, не обнаружит ошибку. Может быть, в кафе сговорились с Лизой обмануть Джона — кто знает? Кафе и Лиза могут, хотя бы на время, заставить Джона поверить, что он получил деньги, которые в действительности не получил.

Джон может предпринять две основные меры, чтобы уменьшить риск быть одураченным полным узлом:

- *Подключиться к нескольким полным узлам одновременно.* Большинство легких кошельков в Биткоин делают это автоматически. Все полные узлы, к которым подключен кошелек Джона, должны принять активное участие в заговоре, чтобы обмануть Джона (рис. 6.31).

Рис. 6.31. Кошелек Джона подключен к нескольким полным узлам. Можно надеяться, что они не сговорились, чтобы обмануть Джона

- *Подключиться к доверенному узлу. Доверенный узел* — это полный узел, действующий на компьютере Джона (рис. 6.32). В этом случае Джон может использовать легкий кошелек на мобильном телефоне для экономии трафика и быть уверенным, что он получает правильную информацию от своего полного узла.

Последний вариант можно использовать, когда есть подозрение, что некоторые полные узлы могут применять правила, с которыми вы не согласны. Единственный способ обрести абсолютную уверенность в следовании нужным правилам — запустить свой собственный полный узел.

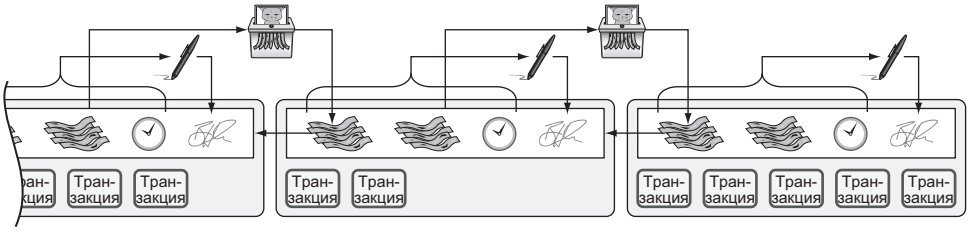
ДОВЕРЕННЫЙ УЗЕЛ

Многие кошельки Биткоин поддерживают возможность подключения к доверенному узлу. Если у вас возникнут затруднения, обратитесь к разработчикам с этим вопросом.

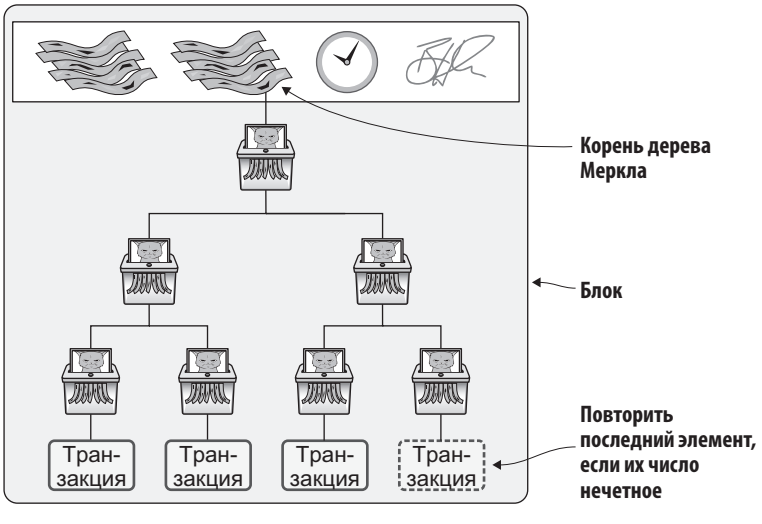
Рис. 6.32. Джон поддерживает свой полный узел, к которому подключен его легкий кошелек

Повторение

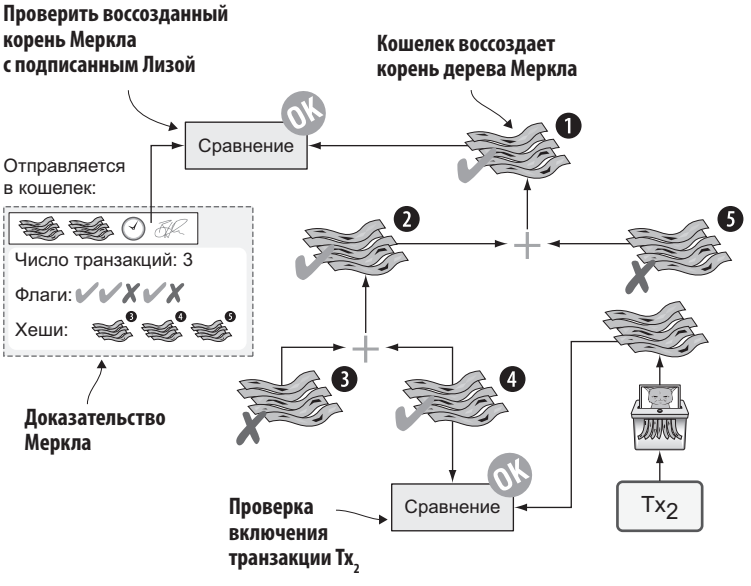
В этой главе вы познакомились с технологией блокчейн и узнали, как она позволяет полным узлам проверить, пыталась ли Лиза удалить или изменить транзакции. Блокчейн — это последовательность блоков, связанных посредством криптографических хешей.



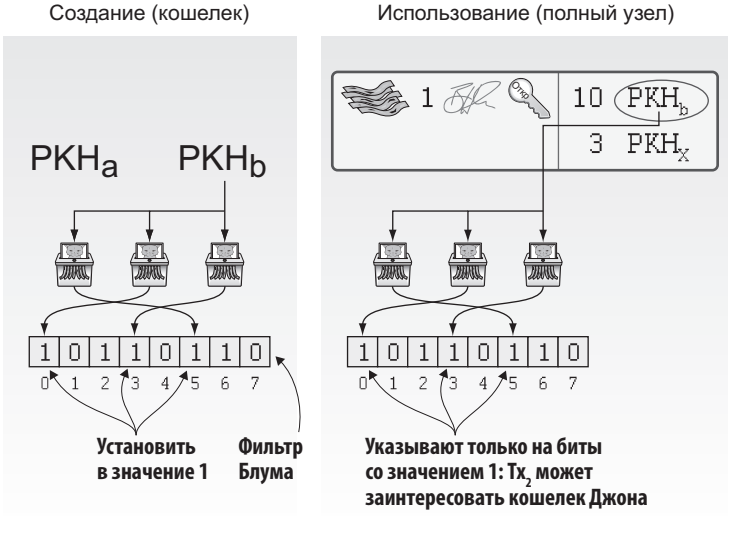
Корень дерева Меркла в заголовке блока — это комбинированный хеш всех транзакций в блоке. Он создается путем хеширования транзакций в структуре дерева Меркла. Чтобы приблизиться на один уровень к корню, хеши объединяются попарно и хешируются.



Полный узел может доказать легкому кошельку, что транзакция включена в блок, отправив доказательство Меркла. Доказательство Меркла состоит из заголовка блока и частичного дерева Меркла. Размер доказательства Меркла растет в логарифмической зависимости от количества транзакций в блоке.



По соображениям конфиденциальности для кошельков желательно получать не только транзакции, которые им действительно интересны. Чтобы скрыть адреса, принадлежащие ему, кошелек использует фильтры Блума, чтобы подписаться на получение большего количества транзакций, чем нужно. Кошелек создает фильтр Блума и отправляет его полному узлу.



Полный узел проверяет разные элементы внутри транзакций — например, PKH в выходах, — используя три хеш-функции. Если в результате хеширования любого элемента получатся индексы, соответствующие единичным битам в фильтре Блума, узел отправит транзакцию в кошелек. Иначе транзакция не будет отправлена.

Эта глава решила проблему с несанкционированным удалением или изменением транзакций. Лиза не может изменить содержимое блокчейна и надеяться, что не будет уличена в мошенничестве.

Однако Лиза все еще может подвергать транзакции цензуре. Она может отказаться подтверждать отправленные ей транзакции. Она имеет абсолютную власть над тем, что будет включено в блокчейн, а что нет. В главе 7 мы затрудним эту возможность.

Изменения в системе

Мы познакомились с технологией блокчейн и заменили электронную таблицу на компьютере Лизы цепочкой блоков (табл. 6.2). В этой главе также была представлена идея общей папки для системы жетонов на булочки. В главе 8 мы заменим эту папку одноранговой сетью полных узлов.

Таблица 6.2. Электронную таблицу заменила цепочка блоков — блокчейн. Мы также добавили общую папку, действующую как замена сети Биткоин

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
~~Электронная таблица~~	~~Блокчейн~~	~~Глава 6~~
Лиза	Майнер	Глава 7
Подпись блока	Доказательство работы	Глава 7
Общая папка	Сеть Биткоин	Глава 8

Наша реализация блокчейна действует подобно блокчейну Биткоин, с одним важным отличием: Лиза подписывает блоки своей цифровой подписью, тогда как в Биткоин они подписываются с использованием доказательства работы.

А теперь выпустим новую версию системы жетонов на булочки. Только взгляните, какие новые замечательные возможности появились (табл. 6.3)!

Таблица 6.3. Примечания к релизу, жетоны на булочки 6.0

Версия	Особенность	Как реализована
НОВАЯ 6.0	Лиза лишена возможности удалять транзакции	Подписи блоков в блокчейне
	Полная проверка узлами	Узлы загружают и проверяют весь блокчейн
	Легкие кошельки экономят трафик	Нечеткие фильтры и доказательство Меркла
5.0	Расходование нескольких «монет» в одном платеже	Несколько входов в транзакции
	Любой может проверить электронную таблицу	Подписи в транзакциях доступны всем
	Отправитель может определять критерии расходования денег	Программы на языке Script внутри транзакции

Упражнения

Для разминки

6.1. Как блоки в блокчейне ссылаются на предшествующие блоки?

6.2. Какая информация используется для вычисления корня дерева Меркла?

6.3. Какая информация используется Лизой при подписании блока?

6.4. Как создаются новые жетоны на булочки (или биткоины)?

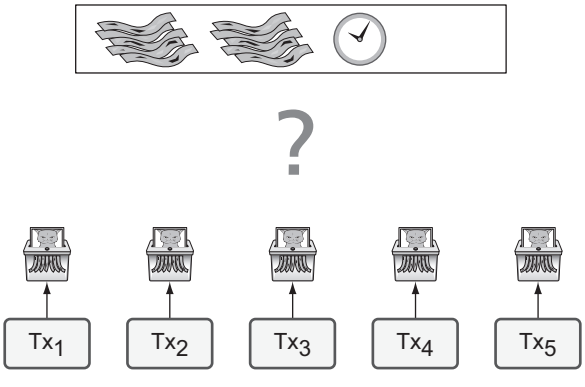
6.5. Какие транзакции будут соответствовать фильтру Блума, состоящему из одних единиц (`1`)?

6.6. Какие элементы транзакции проверяет полный узел, чтобы решить, нужно ли ее отправлять в легкий кошелек или нет? Пропустите это упражнение, если не читали разделы, описывающие работу фильтров Блума.

6.7. Хеш-функции, используемые для создания фильтра Блума, не являются *криптографическими* хеш-функциями. Почему?

Придется пораскинуть мозгами

6.8. Нарисуйте структуру дерева Меркла для блока с пятью транзакциями.



6.9. Лиза подписывает все блоки своим закрытым ключом, специально предназначенным для подписания блоков. Парный ему открытый ключ публикуется в нескольких источниках, таких как внутренняя сеть компании и доска объявлений. Назовите хотя бы одну из угроз безопасности, свойственных этой схеме. Есть две основные угрозы.

6.10. Есть два места, где транзакции или блоки могут подвергаться цензуре. Назовите их.

6.11. Предположим, Лиза создала блок в общей папке, высота которого совпадает с высотой другого уже имеющегося блока. Новый блок содержит те же транзакции, но одна из транзакций в нем заменена другой транзакцией, повторно расходующей одни и те же деньги. Будет ли это обнаружено полным узлом, который:

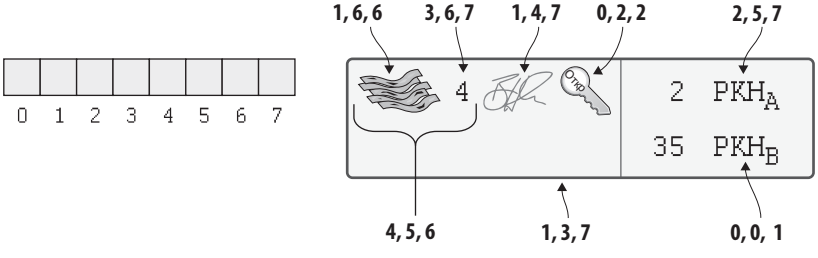
а) еще не загрузил оригинальный блок;

б) уже загрузил оригинальный блок?

ВНИМАНИЕ

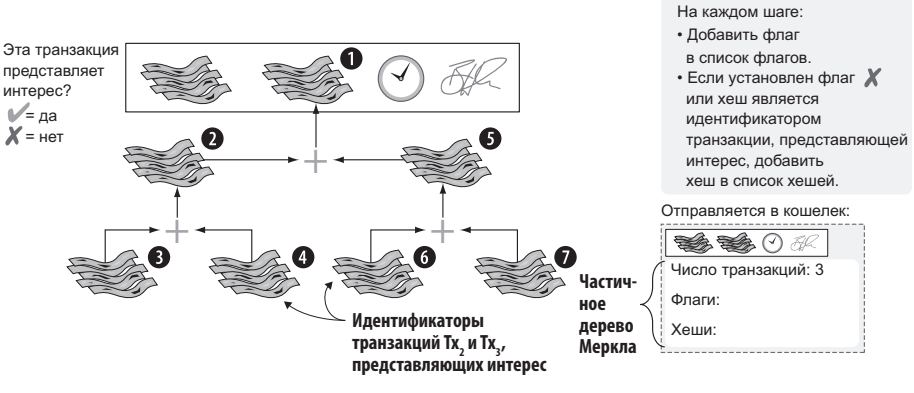
Для решения упражнений 12–15 вы должны прочитать сложные разделы, о которых я предупреждал выше в этой главе.

6.12. Создайте 8-битный фильтр Блума для двух адресов: $@_1$ и $@_2$, где $@_1$ хешируется в индексы 6, 1 и 7, а $@_2$ — в индексы 1, 5 и 7. Допустим, что полный узел использует ваш фильтр, определяя необходимость отправки в кошелек следующей транзакции:

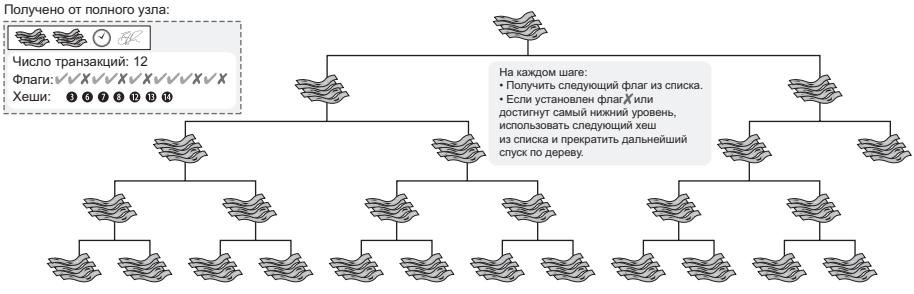


На этом рисунке показаны результаты применения хеш-функций к разным элементам транзакции. Скажите, отправит ли полный узел эту транзакцию в кошелек?

6.13. В разделе «Доказательство включения транзакции в блок» мы сконструировали доказательство Меркла только для одной транзакции, Tx_2. Попробуйте сконструировать частичное дерево Меркла для двух транзакций, Tx_2 и Tx_3. Количество транзакций в блоке равно трем.



6.14. В разделе «Обработка тысяч транзакций в блоке» мы сконструировали частичное дерево Меркла для блока с 12 транзакциями. Какие транзакции полный узел сочтет интересными?



6.15. Представьте, что вы вычислили корень частичного дерева Меркла, как в предыдущем упражнении. Что еще нужно сделать, чтобы убедиться, что в этот блок включена определенная транзакция?

Итоги

- Лиза помещает транзакции в блоки и подписывает их. Это позволяет привлечь ее к ответственности, если она попытается удалить транзакции.
- Подпись фиксирует транзакции в этом и во всех предыдущих блоках, что исключает возможность подделки без повторной подписи мошеннического и всех последующих блоков.
- Транзакции в блоке упорядочиваются в древовидную структуру и хешируются все вместе для создания корня дерева Меркла, который записывается в заголовок блока. Это позволяет создать легкий кошелек.
- Легкие кошельки позволяют экономить трафик, но за счет ухудшения безопасности.
- Безопасность легкого кошелька снижается из-за невозможности выполнить полную проверку транзакций, а также потому, что полный узел может скрыть какие-то транзакции от него.
- Единственный способ добиться абсолютной уверенности в соблюдении всех правил составления блоков — запустить свой полный узел.
- Безопасность легкого кошелька можно улучшить, подключив его к нескольким полным узлам или к доверенному узлу.
- Лиза все еще может подвергать транзакции цензуре.

7 Доказательство работы

Эта глава охватывает следующие темы:

- ✓ противостояние цензуре транзакций добавлением нескольких «Лиз»;
- ✓ состязание за создание следующего блока, или *майнинг*;
- ✓ меры стимулирования майнеров.

В предыдущей главе мы задействовали технологию блокчейн, согласно которой Лиза подписывает все блоки, и тем самым решили проблему несанкционированного удаления или изменения транзакций. В этой главе мы сделаем еще шаг и устраним возможность *цензуры*, чтобы Лиза не могла единолично решать, какие транзакции добавлять в блоки, а какие нет.

Чтобы устранить возможность цензуры, мы заменим цифровые подписи в заголовках блоков на *доказательство работы* (рис. 7.1), допускающее участие в процессе любого количества «Лиз», или *майнеров*. Майнеры будут состязаться за создание следующего блока, пытаясь представить действительное доказательство работы. Получить такое доказательство можно, вычислив огромное количество криптографических хешей. Теперь кошельки смогут отправлять свои транзакции любому или всем майнерам, не опасаясь, что их транзакции не будут обработаны.

С новой системой на основе доказательства работы майнеры будут стремиться создавать блоки как можно меньшего размера, чтобы быстрее выгрузить их в общую папку. У майнеров есть стимул исключать транзакции, чего мы хотели бы избежать. Для стимулирования майнеров к включению транзакции она может предусматривать *комиссионные отчисления*, по-

ступающие майнеру, который произведет блок, подтверждающий транзакцию.

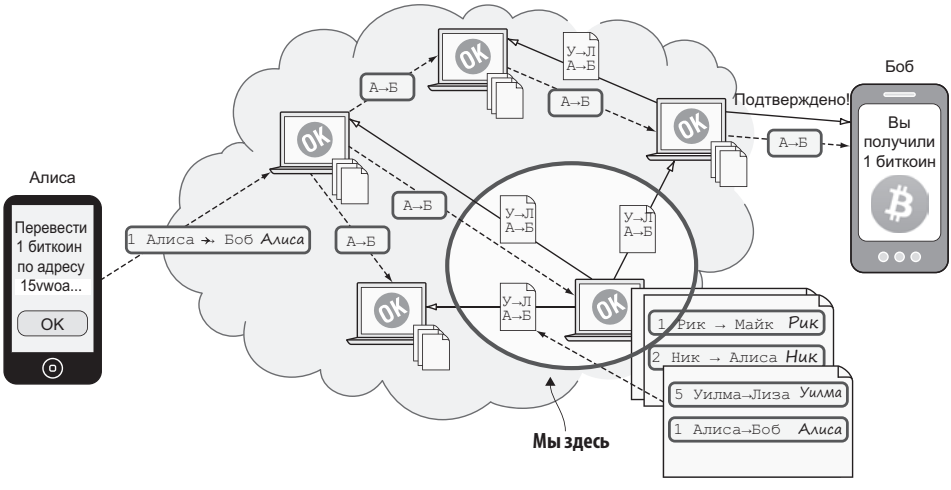


Рис. 7.1. Доказательство работы

Система доказательства работы заменяет цифровые подписи в заголовках блоков. Но цифровые подписи вводились, чтобы помешать Лизе удалять транзакции. Не волнуйтесь, система доказательства работы тоже решает эту проблему, но немного по-другому. Теперь мы не будем доказывать, что Лиза мошенничает, а просто сделаем такое мошенничество слишком сложным и трудоемким.

В этой главе мы обсудим стимулы для майнеров. Как привлечь их заниматься майнингом? Как предотвратить возможность удаления транзакций после подтверждения? Какой вред может нанести майнер, контролирующий большую часть вычислительной мощности, расходуемой на хеширование? Мы обсудим много интересных аспектов, касающихся стимулирования майнеров.

Клонирование Лизы

Мы обсудили некоторые вопросы конфиденциальности в главе 1, в разделах «Проблемы конфиденциальности» и «Децентрализация». Там я отметил, что в системе с централизованным управлением этот руководящий орган обладает абсолютной властью и может единолично решать, кто получит доступ к услуге и для каких целей.

А КАК ЖЕ ОБЩАЯ ПАПКА?

Верно: администратор общей папки тоже является центральным руководящим органом. Он может отказаться добавлять определенные блоки в нее, чтобы никто и никогда их не увидел. Мы исправим этот недостаток в главе 8, когда я представлю одноранговую сеть.

Лиза — как раз такой центральный руководящий орган, и она может подвергать цензуре любые транзакции. Предположим, Лиза только что прочитала книгу известного диетолога, из которой узнала, что булочки вредны для здоровья. Она решила, что должна принять меры против неумеренного потребления булочек в компании, и начала отказываться обрабатывать транзакции, которые, по ее мнению, являются оплатой за булочки, например, выбирая транзакции с выходом 10 CT (рис. 7.2).

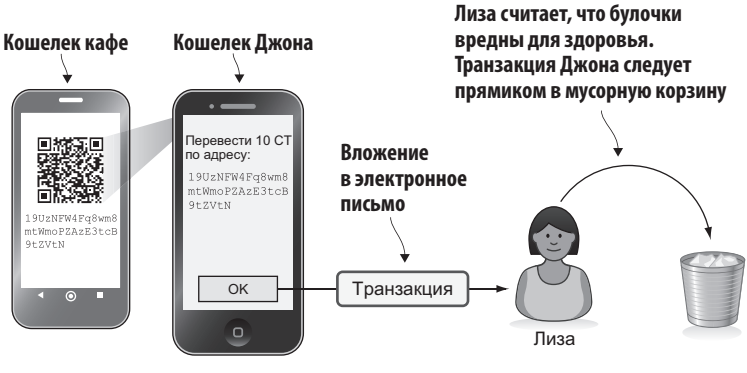


Рис. 7.2. Лиза, действуя как диктатор, может подвергать цензуре транзакции. Никаких булочек Джону!

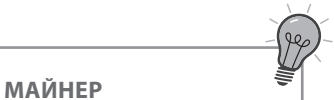
Людям, желающим заплатить за булочку, будет отказано в обслуживании, потому что Лиза не пропустит их платежи. Лиза также может фильтровать другие транзакции, не имеющие ничего общего с оплатой булочек, поскольку подозревает, что они используются для оплаты булочек.

Другая возможная причина цензуры транзакций может заключаться в том, что Acme Insurances угрозами или взятками вынуждает Лизу блокировать подозрительные транзакции, которые могут быть связаны с покупкой булочек, потому что не хочет, чтобы люди страдали от ожирения. Больной человек — это огромные потери для Acme.

А что, если бы у нас было несколько человек, таких как Лиза, чтобы мы могли не полагаться на честность, неподкупность и постоянную доступность кого-то одного? Представьте, что мы разрешили Тому и Ци выполнять ту же работу, что и Лиза. Если кошельки будут отправлять электронные письма с транзакциями всем троим, риск цензуры значительно уменьшится. Но как они будут согласовывать свою работу, чтобы не создавать конфликтующие блоки на одной высоте?

Конфликты между блоками

Предположим, что текущая высота блока равна 100. Том и Ци только что опубликовали свои открытые ключи для подписания блоков на доске объявлений и во внутренней сети компании. Все кошельки начинают отправлять транзакции всем трем производителям блоков, или майнерам. Что случится в этой ситуации, показано на рис. 7.3.



МАЙНЕР

Майнер — это некто, создающий блоки. Лиза — майнер, Том и Ци тоже майнеры.

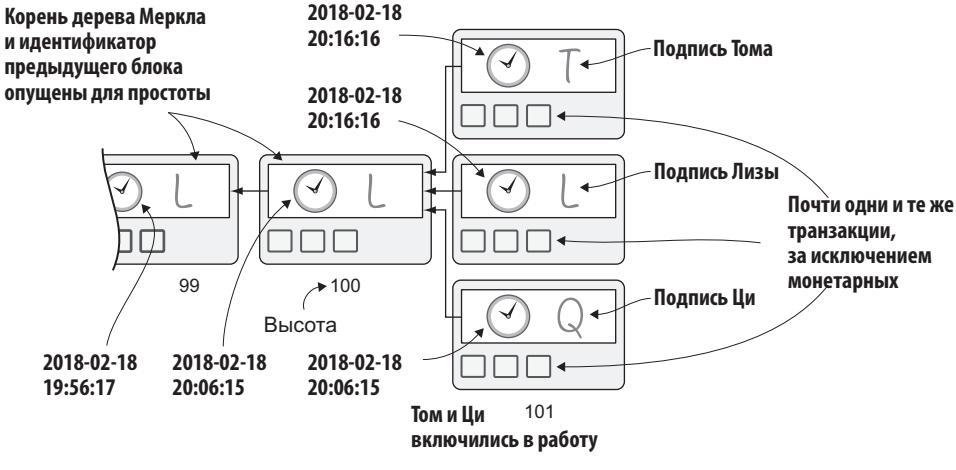


Рис. 7.3. Том и Ци начинают создавать блоки параллельно с Лизой, в результате появляются конфликты между блоками. Заголовки блоков упрощены для ясности

Просто делая все то же, что и Лиза, каждый будет производить блок каждые 10 минут, и в результате получатся три разных блока с почти одинаковыми транзакциями. Главным их отличием друг от друга будут монетарные транзакции и подписи. Монетарные транзакции в блоках Тома будут выплачивать вознаграждение за блок на адрес Тома, а монетарные транзакции в блоках Лизы — на адрес Лизы.

Выбор счастливых чисел

Чтобы предотвратить эту проблему, майнеры должны решить, кто из них создаст следующий блок. Они могут установить очередность, но это сложно, потому что компьютер Лизы может сломаться или Том по какой-то причине решит отказаться создавать блок. В таком случае система остановится.

Попробуем представить другое наивное решение (рис. 7.4). Каждую секунду каждый майнер выбирает случайное число от 0 до 999 999. Если майнеру выпадет число в диапазоне от 0 до 555, он тут же подпишет и опубликует блок. Вероятность выпадения счастливого числа за одну попытку мала — 556/1 000 000, или примерно 1 раз на 1800 попыток. Майнеры выбирают случайное число каждую секунду, поэтому ожидается, что каждому будет выпадать счастливый шанс в среднем раз в 30 минут (1800 секунд). Три майнера вместе будут производить в среднем 1 блок каждые 10 минут.

Когда майнеру выпадает счастливое число, вероятность, что кому-то из двух других майнеров также выпадет счастливое число, невелика. То есть обычно только один майнер сможет произвести следующий блок.

ЧЕСТНЫЕ МАЙНЕРЫ

Это решение я назвал наивным, потому что оно предполагает, что майнеры выбирают случайные числа честно и без подтасовок.

Майнеры сохраняют свои блоки в общей папке, в файлах с именами *<последние-8-шестнадцатеричных-цифр-из-идентификатора-блока>.dat*, поэтому им не приходится волноваться о конфликтах имен файлов для блоков на одной и той же высоте. Примером может служить имя файла 9ce35c25.dat.

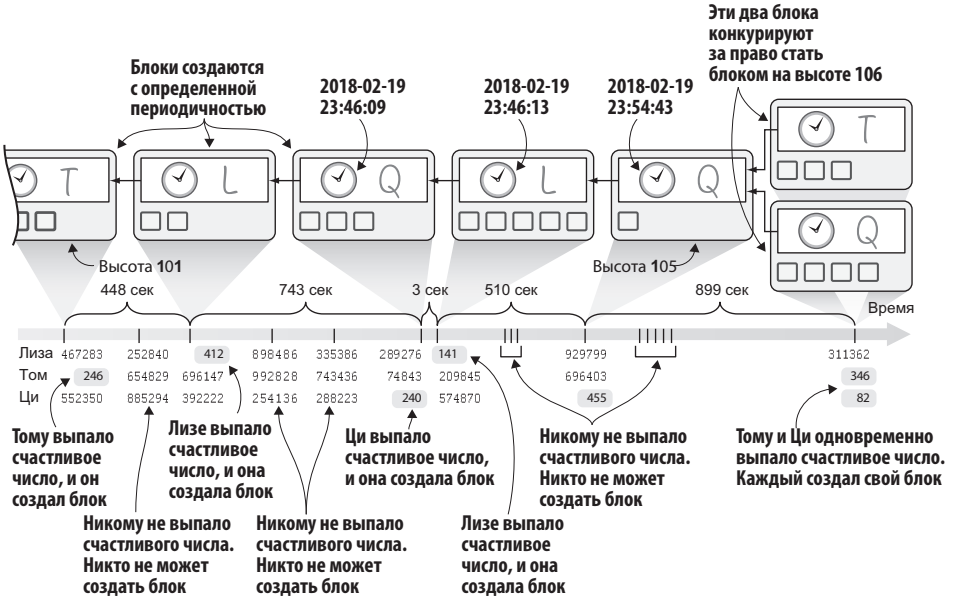


Рис. 7.4. Три майнера занимаются созданием блоков. Обычно конфликтов не возникает, но иногда они все же случаются, как это произошло с блоком на высоте 106

Система работает неплохо, но время от времени двум майнерам одновременно выпадают счастливые числа. При этом никто из них не догадывается, что кому-то другому тоже выпало счастливое число, поэтому оба создают блоки на одной высоте. Эта ситуация называется расщеплением блокчейна, потому что цепочка расщепляется надвое. Обе ветви одинаково действительны, так какая из них «правильная»? Какой майнер «выиграет» и получит награду в 50 CT?

Пока победитель неизвестен. Майнеры должны решить, в какую ветвь продолжить добавлять последующие блоки. Как показано на рис. 7.4, Том и Ци создали свои блоки на высоте 106. Каждый из них, вероятно, мог бы подумать:

- *Том*: я продолжу добавлять к своему блоку, потому что если я выиграю следующий блок, то получу вознаграждение за два блока.
- *Ци*: я продолжу добавлять к своему блоку, потому что если я выиграю следующий блок, то получу вознаграждение за два блока.
- *Лиза*: мне все равно, какой из двух блоков продолжать. Я выберу первый, который успешно проверила: блок Тома. Блоки не могли попасть в общую папку одновременно, поэтому имеет смысл продолжить первый действительный блок, который я увидела.

Когда майнеры выбирают блок на высоте 106 для продолжения, они строят новый блок на высоте 107 и снова начинают выбирать числа. В этой ситуации возможны несколько исходов при условии, что все честны: немедленное разрешение конфликта, отложенное разрешение конфликта или расщепление.

Немедленное разрешение конфликта

В простейшем и наиболее распространенном случае побеждает майнер, которому первому выпало счастливое число. На этот раз повезло Лизе (рис. 7.5).

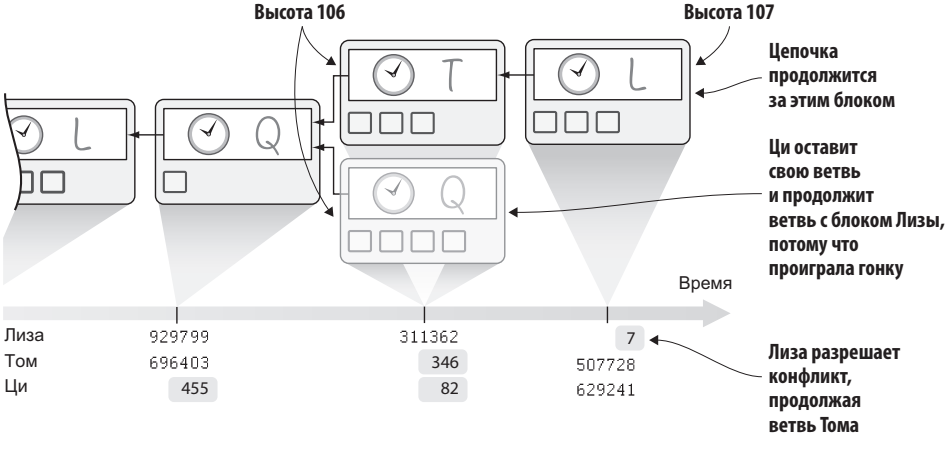


Рис. 7.5. Немедленное разрешение конфликта: Лизе выпало счастливое число

Лиза продолжила строить цепочку за блоком Тома, поэтому ветвь, с которой работают Том и Лиза, оказалась на один блок длиннее. Для этой цепочки блоков действует следующее правило: правильной является самая длинная

ветвь. Это правило изменится далее в главе, но пока мы будем следовать за самой длинной ветвью.

Ци, попытавшаяся начать свою ветвь, замечает, что другая ветвь длиннее, потому что Лиза опубликовала свой блок в той ветви. Ци знает, что все остальные будут выбирать более длинную ветвь, и если она продолжит свою ветвь, то, скорее всего, ее ветвь никогда не догонит другую ветвь и не станет длиннее ее. Поэтому Ци лучше отказаться от своей короткой ветви и перейти к более длинной. Теперь все снова работают над одной ветвью, и конфликт разрешен.

Переключаясь на новую ветвь, Ци помечает все транзакции в своей старой ветви (которые пока отсутствуют в новой ветви) как ожидающие. В будущем они попадут в новые блоки в новой ветви. Узлы поддерживают пул ожидающих транзакций, обычно называемый *пулом памяти*. Пометить транзакцию как ожидающую означает положить ее в пул памяти.

ПРИМЕЧАНИЕ

Множество UTXO конструируется на основе единственной цепочки. Оно не может строиться на основе сразу нескольких ветвей. Полным узлам придется выбирать, за какой ветвью следовать.

Поскольку Ци отказалась продолжать свою ветвь, она также отказалась от награды за блок. Ее блок никогда не станет частью самой длинной цепочки, поэтому она никогда не сможет потратить награду за блок в своем блоке. Только блоки в самой длинной цепочке влияют на содержимое множества UTXO.

Отложенное разрешение конфликта

Но что случится, если в одну и ту же секунду счастливое число выпадет Лизе и Ци (рис. 7.6)? В такой ситуации обе ветви удлинятся на 1 блок каждая. В этом случае нельзя сказать, какая ветка правильная. Майнеры выберут каждый свою сторону и попытаются продолжить свои ветви.

Допустим, следующее счастливое число выпало Тому. Он добавит следующий блок в конец своей ветви, в которой теперь находятся 3 блока, и она станет длиннее другой ветви, в которой всего 2 блока (рис. 7.7).

Все остальные майнеры заметят это и переключатся на ветку Тома. У нас снова есть выигравшая ветвь, и снова Ци не повезло.

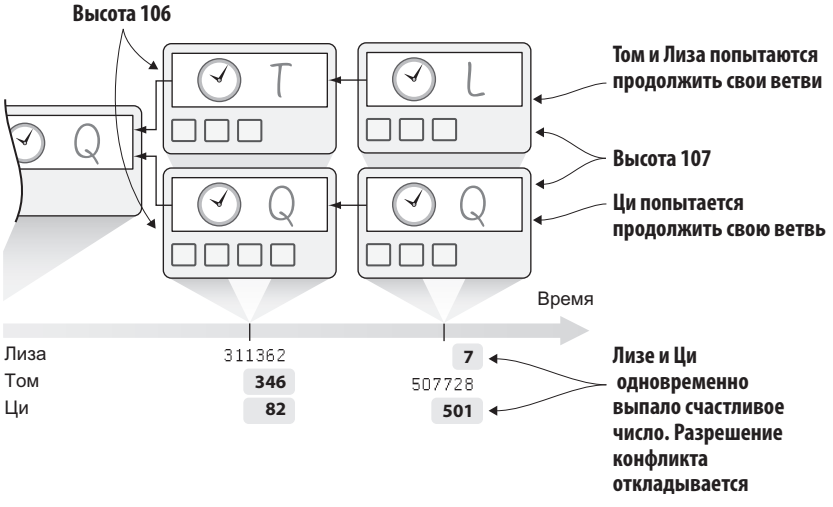


Рис. 7.6. Лизе и Ци выпало случайное число в одну и ту же секунду. Конфликт остается неразрешенным

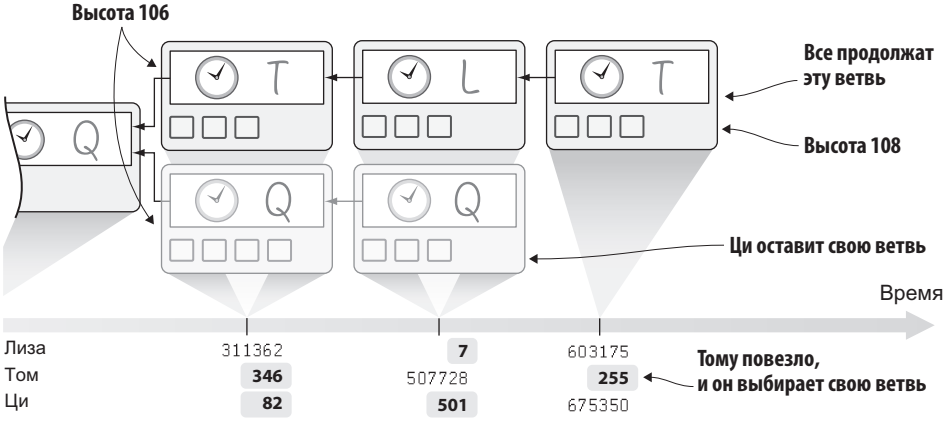


Рис. 7.7. Том — следующий счастливчик, и он продолжает «свою» ветвь, которая теперь становится самой длинной

Повторное расщепление

Теперь представим, что счастливое число одновременно выпало Тому и Лизе. Так как они оба продолжают ветвь Тома, произойдет повторное расщепление (рис. 7.8).

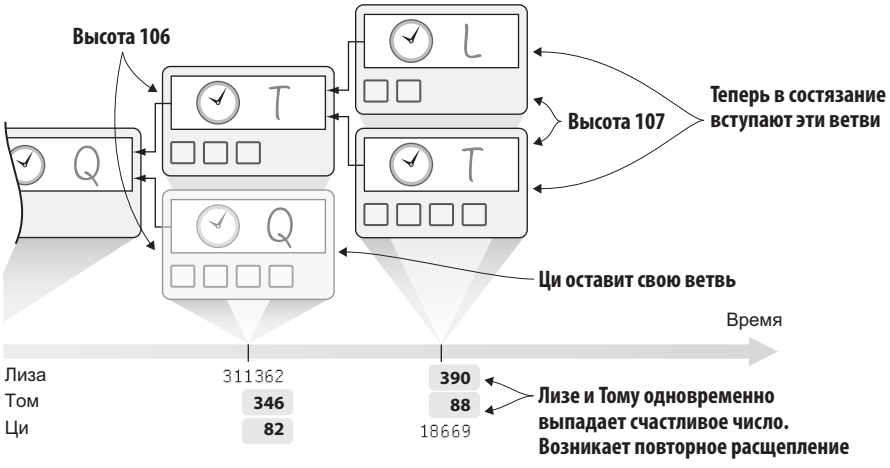


Рис. 7.8. Одна из ветвей снова расщепилась. Этот конфликт разрешается точно так же, как и предыдущий

Теперь у нас есть три ветви. Ветвь Ци, скорее всего, будет заброшена, потому что она короче двух новых ветвей, Лизы и Тома. Этот новый конфликт будет разрешен так же, как и первый:

- немедленно, при добавлении следующего блока;
- с задержкой, если одновременно появятся 2 блока, по одному в каждой ветви;
- когда одна из двух ветвей расщепится надвое.

Вероятность расщепления

В конце концов победит одна из ветвей, возникших после расщепления. Вероятность, что в будущем возникнут две ветви длиной X, быстро уменьшается с увеличением X:

ЭКСПОНЕНЦИАЛЬНАЯ ЗАПИСЬ ЧИСЕЛ

5.6e-4 = 0.00056

2.1e-7 = 0.00000021

*X*e-*Y* — сокращенная форма записи числа $X \times 10^{-Y}$.

Длина ветви	Вероятность	Случается каждые...
1	5.6e-4	2 недели
2	2.1e-7	90 лет
3	7.6e-11	250 000 лет
4	2.8e-14	700 000 000 лет

Лиза (ей 45) почти наверняка столкнется с расщеплением на ветви длиной 1, но ей едва ли придется увидеть расщепление на ветви длиной 2. Но, независимо от длины отщепившихся ветвей, конфликт в конечном итоге будет разрешен. Эта схема выглядит очень неплохо, но у нее есть свои проблемы:

- Майнеры могут лгать, утверждая, что им выпало счастливое число. Мы не сможем доказать, что они получили счастливое число честно.
- С каждым новым майнером система становится более устойчивой к цензуре, но также более уязвимой для кражи закрытых ключей. Чем больше компьютеров с закрытыми ключами, тем выше вероятность кражи ключа. Украв закрытый ключ для подписи блоков, вор сможет создавать блоки, имитируя выпадение счастливого числа и собирая вознаграждение.
- С каждым новым майнером увеличивается риск, что кто-то из майнеров начнет подтасовывать выпадение счастливых чисел.
- Нельзя просто добавлять новых майнеров в систему. С появлением каждого нового майнера нужно уменьшить диапазон счастливых чисел, чтобы сохранить частоту добавления блоков на уровне одного раза в 10 минут и сумму вознаграждения за блок.

РАСЩЕПЛЕНИЯ

Расщепление в Биткоин происходит реже одного раза в месяц, и с течением времени возникают все реже, благодаря более эффективным механизмам проверки и транспортировки.

Очевидно, что в этой системе невозможно бесконтрольно увеличивать число майнеров и придется ограничить их узким кругом доверенных лиц. Когда майнеры начнут мошенничать, вы получите неуправляемый поток блоков, но не сможете уличить их в мошенничестве. Как знать, может, им просто очень и очень повезло.

На чем мы остановились?

В этой главе мы обсуждаем вопрос *доказательства работы*. Я пока не раскрыл полностью значение этого термина, но сделаю это в следующем разделе.

В обзоре системы Биткоин, в разделе «Шаг ❸: Блокчейн» (глава 1), мы видели, что один из майнеров объявляется лидером и решает, какие транзакции и в каком порядке перейдут в следующий блок. Чтобы решить, кто станет лидером, в Биткоин используется доказательство работы (рис. 7.9).

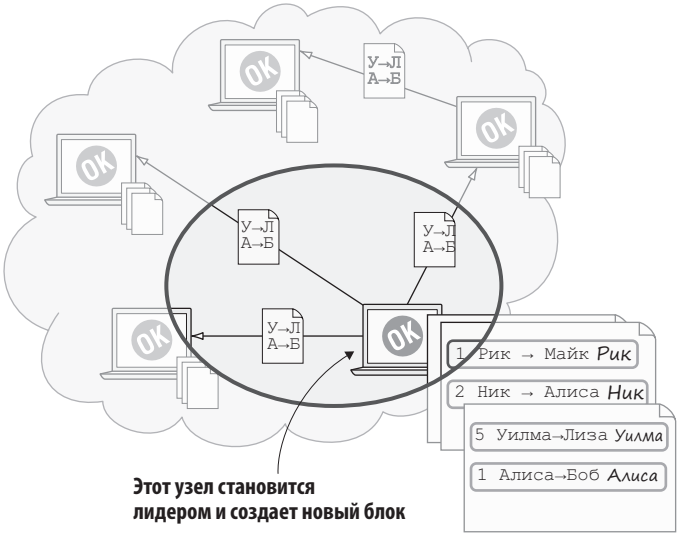


Рис. 7.9. Доказательство работы — это порядок выбора лидера без лидера

Доказательство работы позволяет без участия центрального органа случайным образом выбрать лидера среди майнеров. Продолжая читать эту главу, будьте особенно внимательны, потому что здесь раскрывается суть Биткоина. Именно это делает Биткоин *децентрализованной* системой. Отсутствие единого центра делает систему устойчивой к цензуре. Если система имеет центральный орган, транзакции могут подвергаться цензуре.

Клонирование Лизы — это первый шаг к децентрализации, но пока результат далек от идеала и мы вынуждены доверять майнерам, что те будут честно получать счастливые числа.

Принуждение к честному получению счастливых чисел

Можно ли заставить майнеров честно получать счастливые числа? Оказывается, можно! Можно заставить их произвести огромный объем вычислений на своих компьютерах и доказать, что они выполнили свою работу. Можно заставить их выполнять так много работы, что каждому из трех майнеров потребуется в среднем около 30 минут, чтобы создать блок, и в результате мы получим 10-минутный интервал, как и раньше.

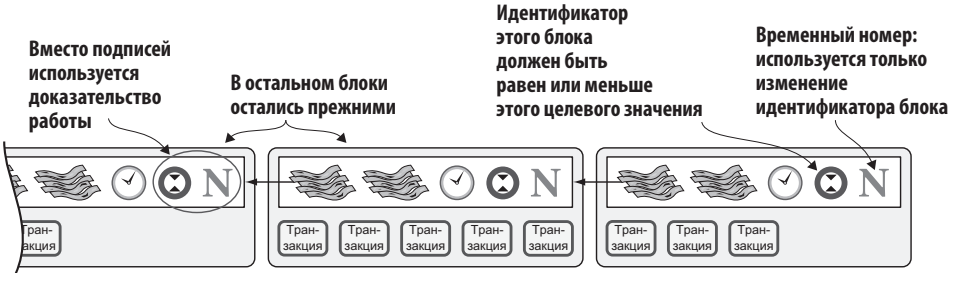
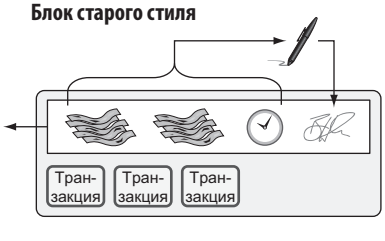


Рис. 7.10. Подписи блоков заменены доказательством работы

Хитрость заключается в замене цифровых подписей в заголовках блоков доказательством работы (рис. 7.10). Предположим, что Ци только что опубликовала блок, и полный узел в кафе проверяет действительность этого блока. Помимо проверки обычных элементов, таких как транзакции и корень дерева Меркла, полный узел должен убедиться, что блок Ци содержит действительное доказательство работы. Доказательство работы действительно, если хеш заголовка блока — идентификатор блока — меньше или равен согласованному целевому значению в заголовке, как показано на рис. 7.11.



Число в заголовке этого блока — `492781982`. Ци подбирает его методом проб и ошибок. В следующем разделе я расскажу, как это делается.

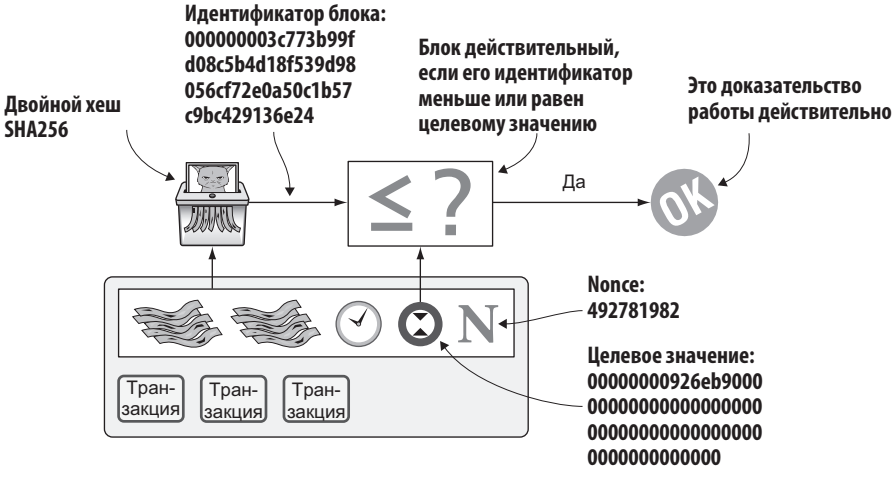


Рис. 7.11. Идентификатор блока должен быть меньше или равен целевому значению в заголовке. Иначе блок будет недействительным

ЦЕЛЕВЫЕ ЗНАЧЕНИЯ В БИТКОИН

Целевое значение записывается в заголовок блока в виде 4 байт, ABCD; 32-байтная цель вычисляется по формуле: BCD $\times$ $2^{8*(A-3)}$. То есть число BCD с A-3 нулевыми байтами после него. Это очень сложно, потому что требуется выразить широкий диапазон целей, 1 – 2^{256}, всего 32 битами. Цель в блоке Ци записывается как `1c926eb9`, что означает число `926eb9` с 25 нулевыми байтами после него (`1c - 3 = 19`, число 25 в шестнадцатеричной форме записи).

Чтобы определить, является ли доказательство работы в заголовке блока действительным, нужно сравнить 256-битный идентификатор блока с 256-битной целью, записанной в заголовке блока. На рис. 7.11 идентификатор блока и цель имеют значения:

```
идентификатор блока: 000000003c773b99fd08c5b4d18f539d98056cf72e0a50c1b57c9bc42913
6e24
цель:                00000000926eb900000000000000000000000000000000000000000000
0000
```

В этом примере идентификатор блока начинается с `000000003...`, а цель — с `000000009...`. Идентификатор блока меньше целевого значения, следовательно, доказательство работы этого блока является действительным.

Цель — это число, согласованное всеми полными узлами и майнерами. Она периодически меняется в соответствии с некоторыми общими правилами.

Это называется *ретаргетингом* (изменением цели), и я опишу этот процесс в следующем разделе. А пока можете рассматривать его как некоторое фиксированное число, которое должно быть установлено в заголовке блока.

Вычисление действительного доказательства работы

Чтобы создать новый блок, майнер должен представить действительное доказательство работы для этого блока. Чтобы получить действительное доказательство работы, майнер должен создать хеш заголовка блока, который меньше или равен целевому значению в заголовке.

Идентификатор блока — это двойной хеш SHA256 заголовка блока. Как рассказывалось в главе 2, единственный способ найти исходный образ криптографической хеш-функции — снова и снова пробовать разные входные данные, пока не будет найдено искомое. То же самое и здесь; майнер должен попробовать разные заголовки блоков, пока не найдет тот, хеш которого будет меньше или равен целевому значению.

Входная строка	Хеш
`Hello1!`	`82642dd9...2e366e64`
`Hello2!`	`493cb8b9...83ba14f8`
`Hello3!`	`90488e86...64530bae`
`...`	`...`

Давайте вернемся немного назад и посмотрим, как Ци создает свой блок. Она создает сам блок, устанавливает целевое значение `00000000926e...` и выбирает одноразовый номер `0`. Затем проверяет, является ли доказательство работы действительным (рис. 7.12).

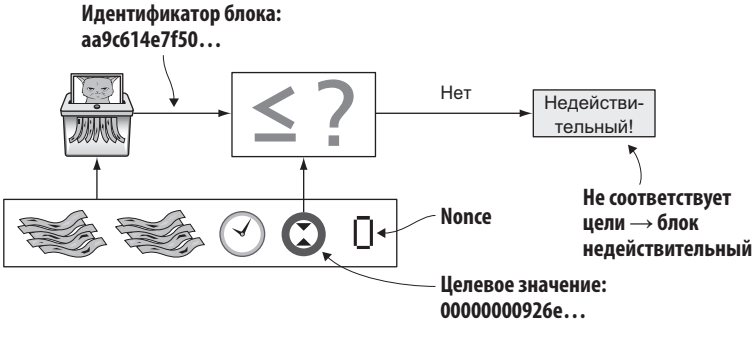


Рис. 7.12. Ци проверяет действительность ее блока с текущим доказательством работы

Она вычисляет идентификатор блока, хешируя заголовок блока двукратным применением хеш-функции SHA256, и получает идентификатор блока `aa9c614e7f50...`. Это число больше целевого:

```
идентификатор блока:  aa9c614e7f5064ef11eedc51856cc7bfcdf71a1f2d319e56d4cc65bda939
be79
цель:                 00000000926eb900000000000000000000000000000000000000000000
000
```

Тогда как правило определения действительности доказательства работы требует, чтобы идентификатор блока был меньше или равен целевому значению. Первая попытка Ци с треском проваливается.

В этот момент в игру включается число nonce[1]. *Nonce* — это самое обычное и ничего не значащее число. Майнер может выбрать для него абсолютно любое значение. Первоначально Ци выбрала число nonce `0`, но с тем же успехом она могла бы выбрать `123` или `92178237`. Число nonce помогает внести изменения, которые повлияют на идентификатор блока, не изменяя каких-либо реальных данных, таких как транзакции или идентификатор предыдущего блока.

NONCE

Nonce — это 32-битное число, поэтому для выбора существует «всего лишь» $2^{32} = 4\ 294\ 967\ 296$ разных временных номеров.

Затем Ци повторяет попытку получить действительное доказательство работы. Она увеличивает nonce до `1` и снова выполняет проверку действительности (рис. 7.13).

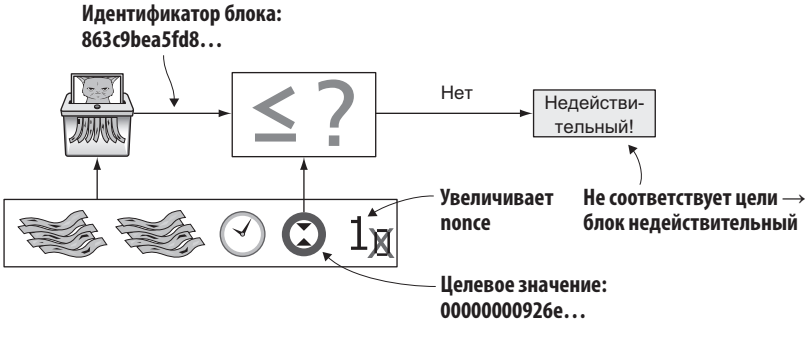


Рис. 7.13. Ци увеличивает nonce, повторяет попытку получить действительное доказательство работы и снова терпит неудачу

[1] Nonce — number that can only be used once. С *англ.* «число, которое может быть использовано лишь однажды». — *Примеч. ред.*

Когда Ци изменяет заголовок блока, увеличивая nonce, меняется и идентификатор блока — любое незначительное изменение в заголовке дает совершенно другой идентификатор блока. Это свойство уже обсуждалось в главе 2, в разделе «Криптографические хеши», когда мы меняли изображение кота (рис. 7.14).

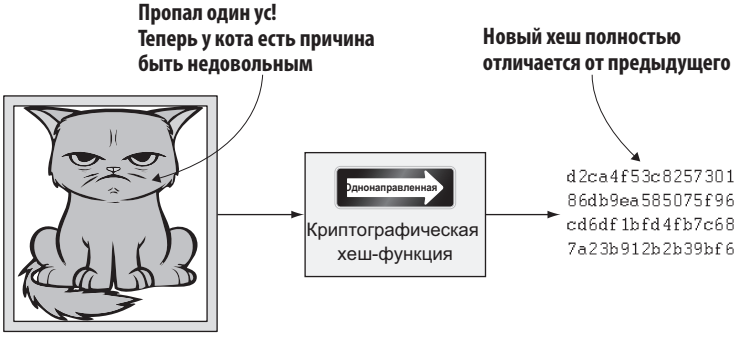


Рис. 7.14. Изменение входного значения криптографической хеш-функции полностью меняет результат

Новый идентификатор блока `863c9bea5fd8...` тоже больше целевого значения. Ци снова терпит неудачу. К сожалению, нет никакого другого способа ускорить решение — Ци должна попробовать еще раз. Она увеличивает nonce до 2 и повторяет проверку (рис. 7.15).

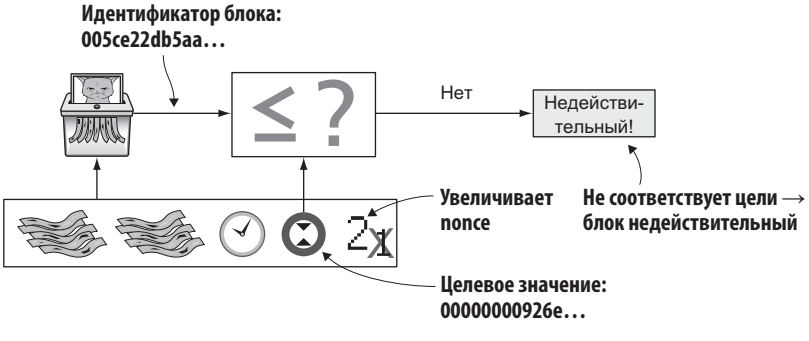


Рис. 7.15. Ци предпринимает третью попытку получить действительное доказательство работы и снова терпит неудачу

Результат тот же: сокрушительный провал. На этот раз получился идентификатор `005ce22db5aa...`, который все еще больше целевого значения.

Она повторяет попытки снова и снова. Для примера на рис. 7.16 показана ее 227 299 125-я попытка. Она была близка к успеху, но близость ничего не значит. Она должна продолжать попытки (рис. 7.17). И наконец, получает результат, показанный на рис. 7.18.

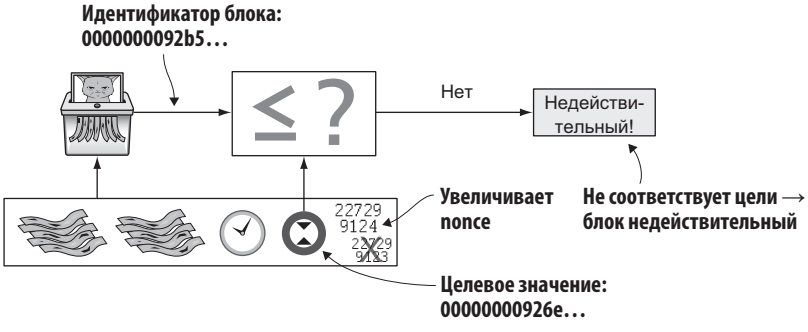


Рис. 7.16. Ци пробует nonce 227 299 124. Близко, но все равно не то!

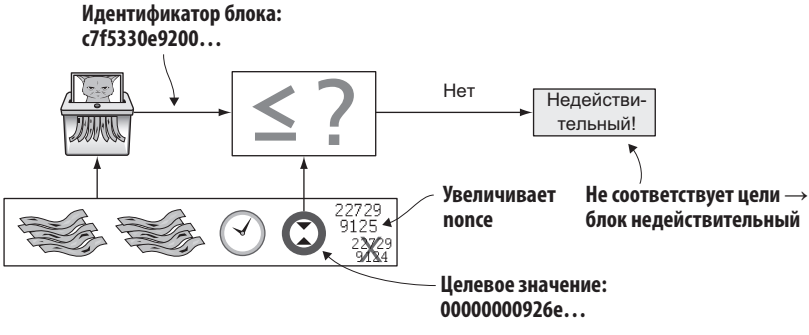


Рис. 7.17. Ци продолжает работать

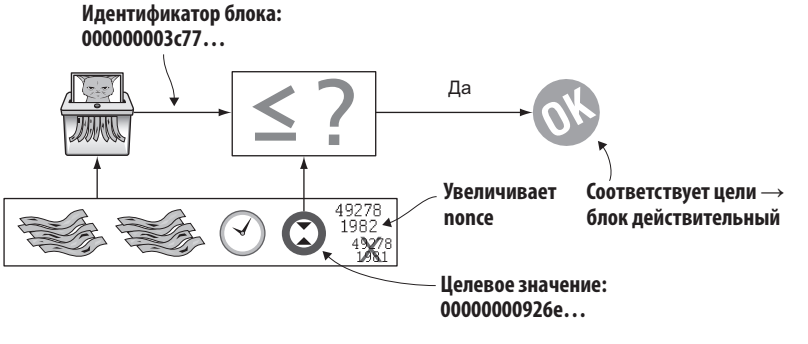


Рис. 7.18. Наконец, nonce 492781982 дает желаемый результат!

Число nonce 492781982 дает в результате идентификатор блока `000000003c77...`. Она сравнивает его с целевым значением:

```
идентификатор блока: 000000003c773b99fd08c5b4d18f539d98056cf72e0a50c1b57c9bc42913
6e24
цель:                0000000926eb900000000000000000000000000000000000000000000000
0000
```

Ура! Этот идентификатор блока меньше целевого значения! Ци проделала большую работу по поиску числа nonce, с которым идентификатор блока оказался меньше целевого значения. Она создала блок с действительным доказательством работы. Отлично, теперь она может опубликовать блок в общей папке.

Важно понимать, что каждый майнер конструирует свой уникальный блок. Например, Том работает над своим блоком параллельно с Ци (и Лизой) и набор транзакций в его блоке отличается от набора транзакций в блоке Ци, потому что его монетарная транзакция выплачивает вознаграждение за блок Тому, а монетарная транзакция в блоке Ци выплачивает вознаграждение Ци. Из-за этого корни дерева Меркла в заголовках блоков будут различаться. Если Том выберет выигрышный nonce Ци (492781982), он, скорее всего, не достигнет цели. Кроме того, их блоки могут отличаться другими элементами, такими как время создания блока или список транзакций.

И что в этом хорошего?

Любой может выбрать блок из общей папки и убедиться, что он соответствует правилам — идентификатор блока меньше или равен согласованному целевому значению. Проверка блока теперь должна выполняться немного иначе (рис. 7.19).

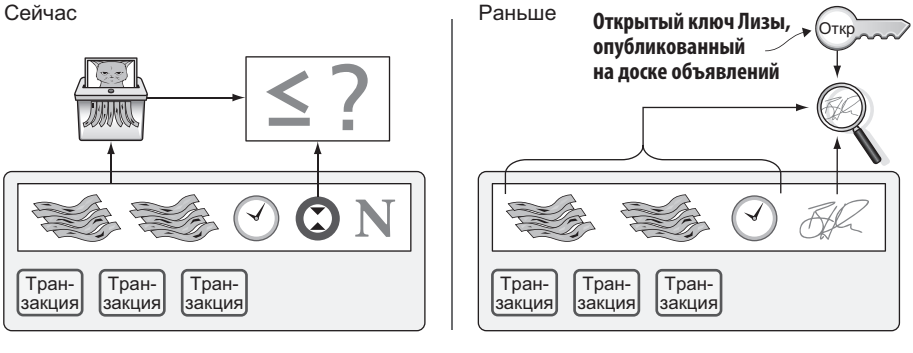


Рис. 7.19. Порядок проверки блока изменился. Теперь проверяющему нужен только сам блок

В отличие от прежнего способа проверки блока с цифровой подписью, теперь полный узел проверяет не действительность цифровой подписи, а действительность доказательства работы.

БЛОКИ СОДЕРЖАТ ВСЕ, ЧТО НУЖНО ДЛЯ ПРОВЕРКИ

Теперь для проверки блока не нужно никакой другой информации, находящейся за пределами блокчейна.

Чтобы проверить действительность блока с использованием доказательства работы, не нужно ничего, кроме самого блокчейна. Раньше для этого требовалась дополнительная информация — открытый ключ майнера, опубликованный, например, на доске объявлений. Это большой шаг вперед к децентрализации. Теперь не осталось централизованных источников открытых ключей, которыми можно манипулировать.

Сравнение со счастливыми числами

Блокчейн будет расти, как и раньше, но теперь на смену вытягиванию счастливых чисел пришло хеширование заголовка блока (рис. 7.20). В табл. 7.1 приводится сравнение двух систем.

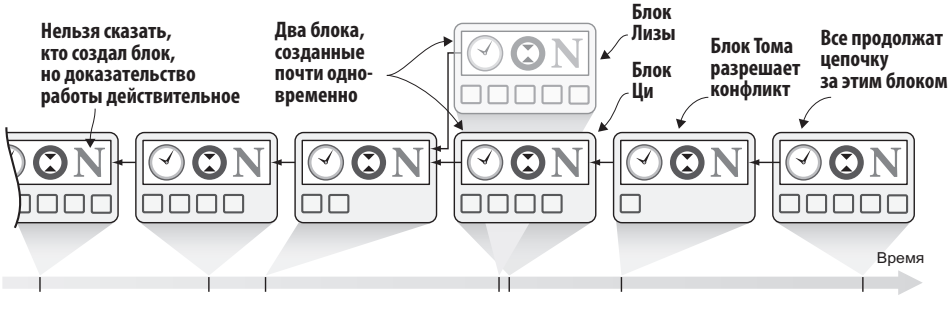


Рис. 7.20. Блокчейн действует точно так же, как в версии с использованием счастливых чисел

Вместо получения случайных чисел каждую секунду, теперь майнеры выбирают новое число примерно каждые 0,02 микросекунды, выполняя криптографическое хеширование. Кроме того, верхняя граница счастливых чисел, или целевое значение, теперь устанавливается равной 256-битному значению 00000000926e... = 926eb9 × 2^{200} вместо 555.

0,02 микросекунды — это лишь пример, сколько времени может уйти на одну «попытку». У разных майнеров это время может отличаться.

Таблица 7.1. Сравнение систем со счастливыми числами и с доказательством работы

Система	Цель	Возможных значений	Выбирается раз в	Среднее время создания одного блока	При расщеплении выбирается
Счастливые числа	555	1 000 000	секунду	10 минут	Самая длинная цепочка
Доказательство работы	$926eb9 \times 2^{200}$	2^{256}	0,02 микросекунды	10 минут	Цепочка с наибольшим объемом работы

Тонкое, но важное отличие состоит в том, что в системе с доказательством работы в случае расщепления выбирается цепочка с *наибольшим накопленным доказательством работы*. В системе со счастливыми числами узлы выбирали самую длинную цепочку. Накопленное доказательство работы — это сумма сложностей отдельных блоков в цепочке.

Сложность определяет, во сколько раз труднее найти действительное доказательство работы для данного блока по сравнению с первичным блоком (genesis block).

Точнее, *сложность блока B* вычисляется по следующей формуле:

$$\frac{\text{целевое значение для первичного блока}}{\text{целевое значение для блока } B} = \frac{(2^{16}-1)\times 256^{26}}{\text{целевое значение для блока } B}.$$

Целевое значение для первичного блока делится на целевое значение для блока *B*, что делает сложность первичного блока равной 1.

Суть в том, что чем выше целевое значение для блока, тем ниже его сложность, и чем ниже целевое значение, тем выше сложность. То есть чтобы получить накопленное доказательство работы, нужно сложить оценки сложности всех блоков в блокчейне.

С этого момента ветвь с наибольшей накопленной работой я буду называть *сильнейшей ветвью*, или *сильнейшей цепочкой*. Иногда такие цепочки называют *лучшими*. Важность различий между длиннейшей и сильнейшей цепочками станет особенно очевидной, когда я познакомлю вас с понятием *корректировки сложности* в разделе «Сила и длина цепочки».

СИЛЬНЕЙШАЯ ЦЕПОЧКА

Сильнейшей называют цепочку с наибольшим накопленным доказательством работы.

Что происходит при исчерпании диапазона числа nonce?

Число nonce — это 32-битное число. Оно довольно мало. Если майнер безуспешно перебрал все 4 294 967 296 возможных чисел, он должен както изменить заголовок блока. Иначе рискует повторить те же самые безуспешные попытки. Изменить заголовок можно несколькими способами (рис. 7.21):

- немного изменить время создания блока;
- добавить, удалить или переупорядочить транзакции;
- изменить монетарную транзакцию.

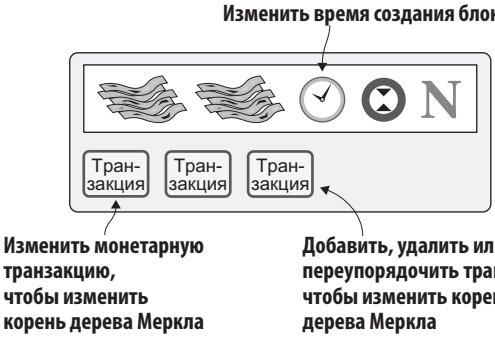


Рис. 7.21. Изменить заголовок блока можно несколькими способами

Время создания блока изменяется легко и просто — достаточно прибавить одну секунду, и заголовок станет другим. Если использовать любой из двух других вариантов, придется пересчитать корень дерева Меркла, поскольку изменятся данные в транзакциях. Когда изменяется корень Меркла, изменяется и заголовок.

После внесения изменений любым из предложенных способов изменится заголовок блока, поэтому nonce можно вновь установить в 0 и начать хеширование с самого начала.

Майнеры должны уйти

Руководство компании считает, что у системы доказательства работы есть преимущества, но оно не хочет платить за электроэнергию, которую расходуют компьютеры, выполняющие все эти сложные вычисления. Компью-

теры работают на электричестве, поэтому чем больше вычислений делает компьютер, тем больше электроэнергии потребляет.

В руководстве решили, что майнеры должны запускать свое программное обеспечение для майнинга за пределами компании, например у себя дома. Это справедливо. В конце концов, майнеры получают 50 CT за каждый найденный блок. Объем электроэнергии, потребленной для производства одного блока, стоит меньше 50 CT. Текущая рыночная стоимость 50 CT равна пяти булочкам в кафе, то есть каждый жетон в настоящее время стоит примерно 20 центов. Каждый блок дает майнеру около 10 долларов в виде жетонов на булочки, что неплохо, учитывая, что каждый из них производит в день около 48 блоков.

Давайте посмотрим на величину *хешрейта* (hashrate) трех наших майнеров. Хешрейт — это количество хешей (попыток), вычисляемых в единицу времени, в данном случае в секунду:

Майнер	Хешрейт (миллионов хешей в секунду)	Ожидаемое число блоков в сутки
Лиза	100	48
Том	100	48
Ци	100	48
Всего	300	144

Эта система будет производить примерно 144 блока в сутки, или один блок каждые 10 минут.

Увеличение хешрейта

Эта система обладает интересной особенностью: майнером может стать *любой желающий* и для этого не требуется ничьего разрешения. Для этого достаточно подготовить компьютер у себя дома и начать создавать блоки. Блоки привязаны не к конкретному человеку, а к количеству вычислительной работы:

- *Лиза наращивает свой хешрейт* — Лиза сочла майнинг на домашнем компьютере выгодным делом и решила установить дома еще один компьютер, что фактически удваивает ее хешрейт.
- *Рашид становится майнером* — Рашид решил заняться майнингом. Он установил дома компьютер и включился в гонку за новыми блоками.

Его компьютер немного быстрее, чем у конкурентов, поэтому он рассчитывает производить больше блоков в день, чем, например, Ци.

После того как Лиза нарастила хешрейт, а Рашид включился в работу, общий хешрейт системы жетонов на булочки значительно увеличился:

ОБЩИЙ ХЕШРЕЙТ В БИТКОИН

На момент написания этой книги общий хешрейт в системе Биткоин составлял примерно 50 эксахешей/с. То есть 50×10^{18} хешей/с.

Майнер	Хешрейт (миллионов хешей в секунду)	Ожидаемое число блоков в сутки
Лиза	200	96
Том	100	48
Ци	100	48
Рашид	150	72
Всего	550	264

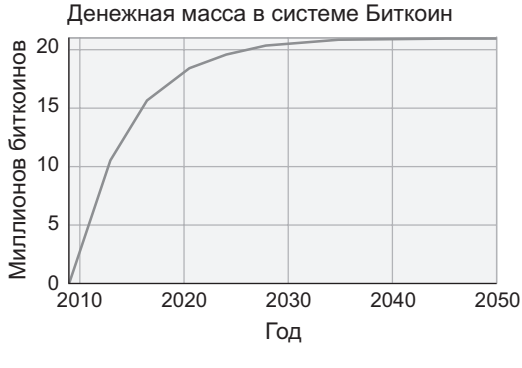
Посмотрите: мы производим в сутки больше блоков, чем планировали! Цель — 144 блока в день, а число 264 намного больше целевого значения. *Скорость создания блоков* слишком высока, она почти вдвое превышает желаемую.

Проблемы высокой скорости создания блоков

Высокая скорость создания блоков может показаться выгодной, потому что это повлечет уменьшение времени подтверждения транзакций, но она влечет за собой несколько проблем.

Слишком большая скорость создания денег

Вспомните планы эмиссии денег, приводившиеся в главе 2. Согласно плану, в течение первых четырех лет предполагалось эмитировать половину денежной массы: 10,5 млн СТ; затем в течение следующих четырех лет предполагалось эмитировать половину этой половины:



5,25 млн CT; и так далее, пока объем эмиссии не округлится до 0. Весь этот процесс займет около 131 года.

Но теперь, когда Лиза нарастила свои вычислительные мощности и в работу системы включился Рашид со своим компьютером для майнинга, эмиссия ускорилась. При такой высокой скорости создания блоков все жетоны на булочки будут эмитированы за вдвое меньший срок.

То есть темп прироста денежной массы в 264/144 = 1,8 раза превышает желаемый.

Больше расщеплений

Расщепление — естественное явление, и оно происходит время от времени. Но с увеличением скорости создания блоков увеличивается и вероятность расщепления. Представьте, что 3000 человек начали заниматься майнингом, установив компьютеры у себя дома. Это увеличит скорость создания блоков в 1000 раз. Каждую секунду несколько майнеров будут находить действительное доказательство работы и публиковать блок. Расщепление в такой ситуации будет происходить почти на каждой высоте. Это сделает транзакции в последних блоках менее надежными, потому что эти блоки легче отделить от основной цепочки.

Это также может привести к ухудшению безопасности, поскольку, если на каждую из двух ветвей придется примерно по 50% от общего хешрейта, безопасность каждой отдельной ветви уменьшится вдвое. Подробнее о безопасности блокчейна мы расскажем в разделе «Какой вред могут принести майнеры?».

Как это исправить?

Мы решили сложную проблему «честного выпадения счастливого числа» интересным способом. Давайте посмотрим, какие еще проблемы остались из перечисленных в разделе «Вероятность расщепления»:

- ☑ Майнеры могут лгать, утверждая, что им выпало счастливое число. Мы не сможем доказать, что они получили счастливое число честно.
- ☑ С каждым новым майнером система становится более устойчивой к цензуре, но также более уязвимой для кражи закрытых ключей. Чем больше компьютеров с закрытыми ключами, тем выше вероятность кражи ключа. Украв закрытый ключ для подписи блоков, вор сможет

создавать блоки, имитируя выпадение счастливого числа и собирая вознаграждение.

- ☑ С каждым новым майнером увеличивается риск, что кто-то из майнеров начнет подтасовывать выпадение счастливых чисел.
- ☐ Нельзя просто добавлять новых майнеров в систему. С появлением каждого нового майнера нужно уменьшить диапазон счастливых чисел, чтобы сохранить частоту добавления блоков на уровне одного раза в 10 минут, и сумму вознаграждения за блок.

Нерешенной осталась всего одна проблема. Мы исправим ее в следующем разделе.

Корректировка сложности

Теперь, когда в системе появилось больше майнеров и увеличился общий хешрейт, увеличилась скорость создания блоков. Это связано с тем, что вместе майнеры делают больше попыток в секунду, чем раньше, что привело к увеличению количества блоков, создаваемых в час.

Майнеры согласовали целевое значение в заголовке блока, и теперь нам нужно скорректировать сложность майнинга, чтобы увеличить или уменьшить общую скорость хеширования. Целевое значение изменяется через каждые 2016 блоков. Эта корректировка называется *корректировкой сложности*, а величина 2016 называется *периодом корректировки*. Напомню, что каждый блок содержит монетарную транзакцию, которая создает 50 новых жетонов на булочки. Для поддержания плановой эмиссии жетонов нам нужно, чтобы в среднем блоки создавались со скоростью 1 блок в 10 минут. То есть 2016 блоков должны создаваться не быстрее чем за две недели.

Если последний период корректировки длился более двух недель, целевое значение следует увеличить, чтобы увеличить вероятность обнаружения соответствующего хеша заголовка блока и тем самым уменьшить сложность. Если период корректировки длился меньше двух недель, целевое значение следует уменьшить, чтобы уменьшить вероятность его достижения, и тем самым увеличить сложность.

Новое целевое значение, N, вычисляется как $N = O \times F$, где O — старое целевое значение, а F — коэффициент изменения целевого значения, который зависит от продолжительности последнего периода корректировки, как показано на рис. 7.22.

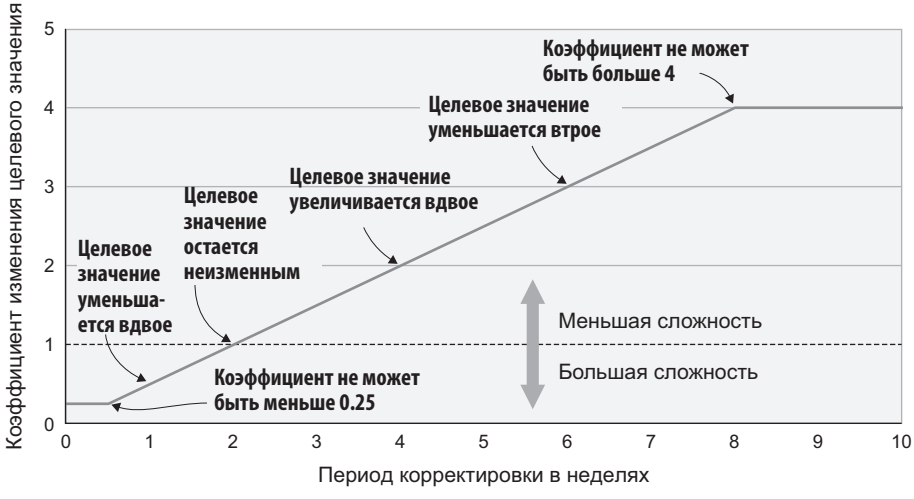


Рис. 7.22. Корректировка сложности в зависимости от продолжительности создания последних 2016 блоков. Цель — ограничить скорость 2016 блоками в две недели

В общем случае новое целевое значение N вычисляется на основе O и протяженности T последнего периода корректировки, как показано ниже:

$$N = O * \begin{cases} \frac{1}{4} & \text{если } T < 0,5 \\ \frac{T}{2} & \text{если } 0,5 \le T \le 8. \\ 4 & \text{если } 8 < T \end{cases}$$

Новое целевое значение не может быть больше или меньше старого более чем в 4 раза, чтобы ограничить эффект некоторых атак двойного расходования, когда кто-то изолирует узел жертвы от честных узлов, чтобы манипулировать сложностью в свою пользу. Вы можете прочитать об этом на веб-ресурсе 15 (приложение B).

Правила для отметок времени

Заголовок блока содержит *отметку времени*. Отметки времени важны для автоматической настройки целевого значения без вмешательства человека, чтобы обеспечить среднюю скорость создания 1 блока за 10 минут. Скорость

создания блоков, в свою очередь, важна для получения предсказуемой эмиссии новых жетонов на булочки.

ПРИМЕЧАНИЕ

Отметки времени также используются в транзакциях. Подробнее об этом я расскажу в главе 9.

Майнер, создающий блок, устанавливает отметку равной текущему времени и потом пытается получить доказательство работы. Но поскольку разные полные узлы работают на разных компьютерах, их часы могут быть синхронизированы неидеально.

Предположим, что Лиза создает и публикует блок с отметкой времени 2017-08-13 07:33:21 UTC. Затем Том создает следующий блок, но его часы отстают от часов Лизы.

Отметка времени в блоке Тома показывает время, предшествующее отметке времени в предыдущем блоке. В этом нет ничего страшного, если отметки времени отличаются несильно (рис. 7.23).

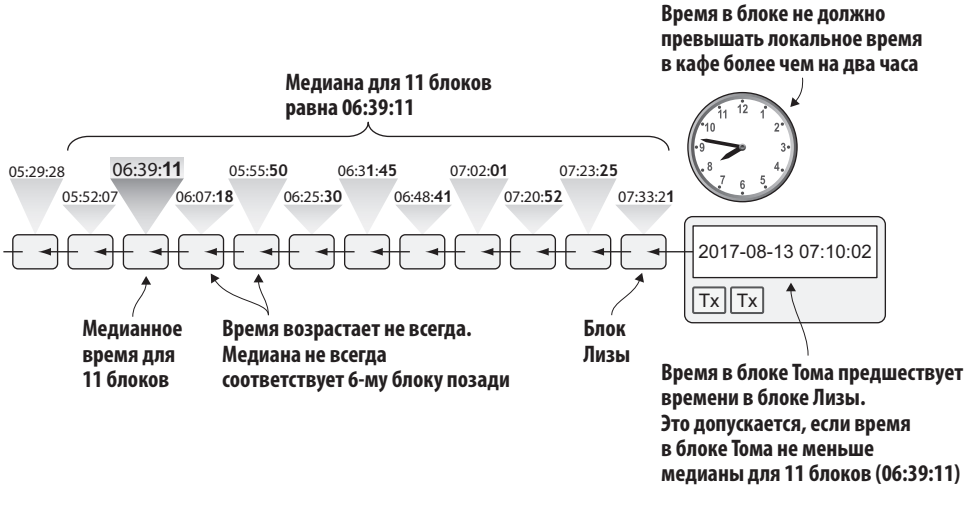


Рис. 7.23. Два блока с отметками времени в убывающем порядке. Это нормально

Отметка времени должна соответствовать нескольким правилам. Предположим, что полный узел кафе проверяет блок Тома:

- Время в блоке обязательно должно быть позже медианного времени для 11 предыдущих блоков. Обычно эту медиану называют *медианой предыдущего периода*.
- Время в блоке не должно превышать текущее время в кафе более чем на два часа.

Эти правила гарантируют, что никто не сможет манипулировать отметками времени в своих блоках, чтобы как-то повлиять на вычисление следующего целевого значения. Представьте, что последний блок перед изменением целевого значения имел отметку времени, отстоящую от текущего времени на шесть недель в будущем. Это приведет к увеличению следующей цели в 4 раза, как показано в табл. 7.2.

Таблица 7.2. Майнер-злоумышленник сфальсифицировал отметку времени в последнем блоке из 2016 перед корректировкой. H — высота первого блока в периоде корректировки. Новое целевое значение увеличится в 4 раза

Высота блока	Отметка времени (без секунд)	Время от начала периода
H	2017-07-31 06:31	0
H + 1	2017-07-31 06:42	11:17
...	...	...
H + 2 013	2017-08-14 07:22	2 недели и 51 минута
H + 2 014	2017-08-14 07:33	2 недели и 1 час 2 минуты
H + 2 015	2017-09-25 08:51	8 недель и 2 часа 20 минут

Последняя отметка времени на шесть недель превышает фактическое время создания блока. Все полные узлы отклонят этот блок, потому что он нарушает правила отметки времени. Кто-то предпринял попытку манипулирования целью. Если бы этот блок был принят, следующее целевое значение увеличилось бы в четыре раза, что в четыре раза облегчило бы поиск действительного доказательства работы. Такое манипулирование запрещено только что описанными правилами. Благодаря невозможности «опередить время» более чем на два часа, майнеры не смогут существенно повлиять на выбор следующего целевого значения.

Сила и длина цепочки

А теперь вернемся к обсуждению силы цепочки и поговорим, почему важно учитывать не только длину цепочки. Очевидно, что чем сложнее переписать

историю цепочки, тем лучше, поэтому мы должны следовать за сильнейшей цепочкой. Но в каких случаях сильнейшая и длиннейшая цепочки оказываются разными цепочками?

Вот эти основные случаи:

- естественное расщепление непосредственно перед корректировкой;
- непреднамеренное расщепление из-за несовместимости версий программного обеспечения;
- умышленное расщепление, как атака на честную цепочку.

Рассмотрим только первый из них. Представим, что возникло естественное расщепление (рис. 7.24).

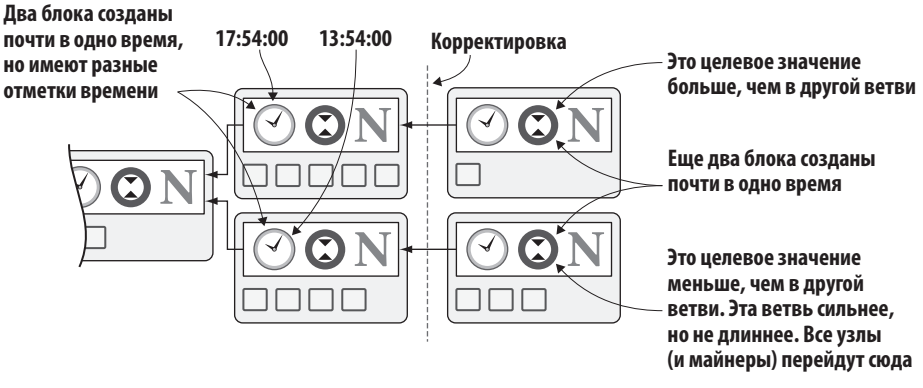


Рис. 7.24. Естественное расщепление с разными отметками времени в разных ветвях может сделать одну ветвь сильнее другой при корректировке

Это маловероятный сценарий, но мы должны рассмотреть его, потому что он *может* произойти. Расщепление происходит непосредственно перед корректировкой, и отметки времени в двух блоках отличаются на четыре часа. Затем одновременно создаются еще два блока, по одному в каждой ветви. Эти новые блоки построены на основе разных историй. Отметки времени в блоках, предшествовавших корректировке, отличаются на четыре часа, из-за чего в ветвях вычисляются разные целевые значения. Напомню формулу корректировки:

ОТМЕТКИ ВРЕМЕНИ

Отметки времени не могут забегать вперед более чем на два часа.

$$N = O^* \begin{cases} \frac{1}{4} & \text{если } T < 0,5 \\ \frac{T}{2} & \text{если } 0,5 \le T \le 8. \\ 4 & \text{если } 8 < T \end{cases}$$

Поскольку новые целевые значения отличаются, сложности последних блоков в ветвях тоже будут отличаться, а значит, цепочки будут иметь разную силу, потому что теперь ветви имеют разные накопленные доказательства работы.

Какой вред могут принести майнеры?

В шестой главе мы лишили Лизу возможности отменять транзакции и избавились от необходимости доказывать факт ее мошенничества. Для этого мы потребовали от Лизы добавлять цифровую подпись в блоки, чтобы каждый мог убедиться, что Лиза одобрила их. Если позже она подпишет альтернативный блок на той же высоте, в котором заменит свою транзакцию, другой, выплачивающей деньги ей самой, все заметят это и привлекут ее к ответу.

Сейчас ситуация изменилась. Лиза больше не подписывает блоки. Блоки анонимны — ничто не связывает Лизу с конкретным блоком. Разве это не значит, что она снова может потратить вдвое больше?

Да, если ей очень повезет.

Двойное расходование

Предположим, Лиза решила купить булочку в кафе. Но к моменту платежа она подготовила транзакцию, расходующую те же деньги (рис. 7.25).

C — это транзакция перевода денег в кафе. **L** — транзакция, повторно расходующая те же средства, которую Лиза собирается использовать, чтобы вернуть себе деньги. В отдельности обе транзакции полностью действительные, но они не могут быть действительными одновременно, поскольку обе расходуют один и тот же выход. Выход можно израсходовать только один раз.

Лиза отправляет честную транзакцию **C** всем майнерам. Пока другие майнеры пытаются добавить честную транзакцию в блок и создать действительное

доказательство работы, Лиза втайне помещает транзакцию **L**, повторно расходующую те же средства, в свой секретный блок и начинает работать над этим блоком (рис. 7.26).

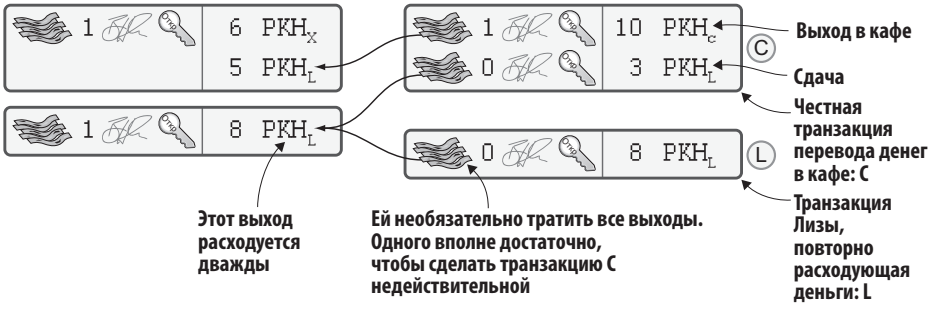


Рис. 7.25. Лиза создала две транзакции, расходующие один общий выход

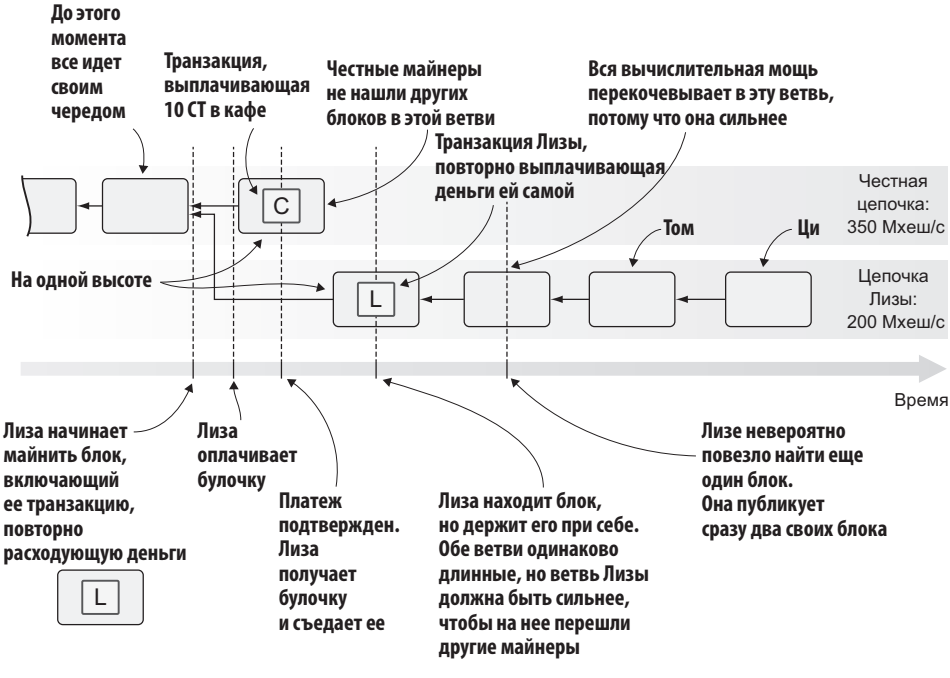


Рис. 7.26. Лиза совершает атаку двойного расходования средств, и ей это удается, несмотря на то что имеет невысокий хешрейт

Лиза стремится тайно найти действительное доказательство работы для своей мошеннической ветви, содержащей **L**, которое превысит доказательство

работы в честной цепочке. Если ей это удастся, она опубликует все блоки в своей ветви и все майнеры переключатся на ее ветвь и продолжат наращивать ее. Для простоты предположим, что все описанное происходит без корректировки сложности — мы находимся в середине периода корректировки, а значит, все блоки имеют одинаковое целевое значение (сложность), поэтому мы можем ограничиться наблюдением за длиной ветви, а не за ее силой (накопленным доказательством работы).

Пока майнеры пытаются подтвердить честную транзакцию **C** Лизы, сама Лиза работает над поиском действительного доказательства работы для своего блока с транзакцией **L**, повторно расходующей те же деньги. Кафе ждет появления действительной транзакции, прежде чем выдать булочку.

В конце концов, честная транзакция будет подтверждена в честной цепочке. Кафе видит этот блок, проверяет его и выдает булочку Лизе. Лиза съедает ее. В то время когда она проглатывает последнюю крошку, компьютер Лизы находит действительное доказательство работы для ее блока. Она не спешит публиковать свой блок, потому что это ей не поможет. Майнеры уже занимаются майнингом в честной ветви, потому что именно там они впервые увидели блок.

Суммарный хешрейт всех майнеров в честной цепочке составляет 350 Мхеш/с, тогда как хешрейт Лизы составляет всего 200 Мхеш/с. Это означает, что блоки в честной цепочке должны обнаруживаться чаще, чем в мошеннической цепочке Лизы.

Но иногда кому-то очень везет. Повезло и Лизе — она нашла еще один блок для своей мошеннической ветви. Теперь у нее есть 2 блока, тогда как в честной ветви только 1 блок. В ветви Лизы больше доказательств работы, чем у честных майнеров в их ветви. Лиза публикует свои 2 блока в общей папке.

ЗА КАКОЙ ВЕТВЬЮ ПОСЛЕДОВАТЬ?

Майнер не обязан следовать за первым увиденным блоком. Но программное обеспечение Bitcoin Core, наиболее широко используемое в системе Биткоин, следует за первым увиденным блоком.

Другие майнеры увидят эти 2 блока, заметят, что ветвь Лизы содержит больше доказательств работы, чем честная ветвь, и переключатся на ветвь Лизы. Переключающиеся майнеры не могут видеть, что совершается преступление, и не знают, кто создает блоки; они просто перейдут в сильнейшую действительную цепочку.

В результате транзакция **C**, переводящая деньги в кафе, фактически будет отменена. Она больше не является частью цепочки с наибольшим доказательством работы. Кафе потеряло 10 CT, которые, как думали его сотрудники, были получены, когда выдавали булочку Лизе.

С этого момента новые блоки будут добавляться в ветвь Лизы, и все продолжится как обычно. Блок с транзакцией **C** потеряет актуальность.

Защита от атак двойного расходования

Хотя все шансы против Лизы, ей *может* повезти в атаке с двойным расходованием, как было показано выше. Попытка дважды потратить 10 CT экономически нецелесообразна с точки зрения Лизы. Она рискует потратить много электроэнергии и в конечном итоге получить неактуальные блоки, если ее попытка не увенчается успехом. Она недополучит вознаграждение за эти неактуальные блоки.

Но что, если она попытается удвоить расходы, превышающие 10 CT, скажем, 100 000 CT? Тогда Лизе стоит попытаться удвоить расходы. Только представьте, она может купить все кафе и успешно провести атаку двойного расходования. После этого она получит кафе и по-прежнему будет иметь 100 000 CT.

Владелец кафе готов продать его Лизе за 100 000 CT. Но он, конечно же, знает об атаках двойного расходования. Итак, владелец кафе говорит Лизе, что продаст ей кафе за эти большие деньги после шести подтверждений.

Что это значит? Лиза должна заплатить владельцу кафе 100 000 CT и дождаться, пока транзакция будет включена в блок и после этого блока будет построено еще 5 блоков. Только после этого владелец передаст кафе Лизе.

Чтобы провести атаку двойного расходования в этих условиях, Лиза должна тайно построить альтернативную ветку, как и в предыдущей атаке, пока владелец кафе ожидает появления шести подтверждений. Когда он увидит шесть подтверждений и передаст кафе Лизе, она должна в какой-то момент выгрузить в общую папку более сильную ветку с повторным расходом. Это значит, что Лизе должно везти дольше, чем в предыдущем примере.

Давайте посмотрим, как мог бы развиваться этот сценарий (рис. 7.27).

Как и ожидалось, в долгосрочной перспективе Лиза не смогла произвести больше блоков, чем честные майнеры. Она сдалась со счетом 7–4.

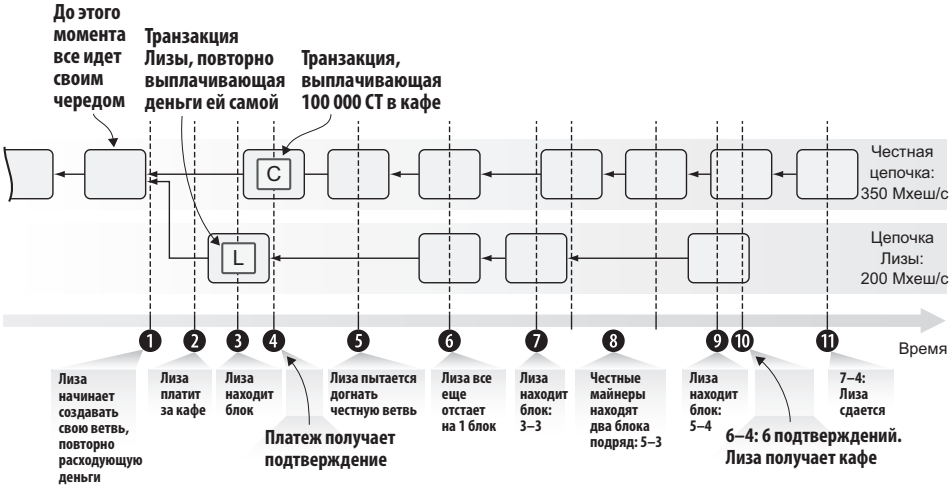


Рис. 7.27. Лиза пытается создать шесть подтверждений для транзакции, повторно расходующей те же деньги, и терпит неудачу

Последовательность событий, происходящих в этом примере, представлена в следующей таблице:

Событие	Счет (C–L)	Комментарий
❶, ❷	0–0	Лиза начинает майнить свою тайную ветвь с транзакцией, повторно расходующей деньги. Она также отправляет платеж честным майнерам
❸	0–1	Лиза находит блок, но держит его в секрете. Она не хочет, чтобы кафе заметило, что происходит атака двойного расходования
❹	1–1	Честная оплата, C, получает первое подтверждение. Кафе ждет появления еще 5 блоков, прежде чем выполнить свою часть сделки
❺, ❻, ❼, ❽, ❾	5–4	Лиза продвигается вперед, но она отстает на 1 блок и, чтобы добиться успеха, должна создать на 2 блока больше, чем кафе
❿	6–4	Честная сделка находит шесть подтверждений. Лиза получает кафе. Передаточный акт подписан. Лиза продолжает пытаться наверстать отставание
⓫	7–4	Лиза понимает, что не догонит. Вероятность создать на 4 блока больше, чем группа честных майнеров, ничтожна

Лиза сдалась по нескольким причинам:

- Она поняла, что ей не хватает хешрейта, чтобы догнать и превзойти честную цепочку. В каждый конкретный момент вероятность, что Лиза найдет следующий блок, равна 200/550 = 0,36. А вероятность, что честные майнеры найдут следующий блок, составляет 1 – 0,36 = 0,64. Блоки в честной цепочке будут создаваться намного быстрее.
- Все время, пока она продолжает попытки, ее компьютер потребляет электроэнергию, которая стоит денег. Если она не добьется успеха, затраты на электроэнергию окажутся напрасными.
- За каждый блок, добываемый ею в своей цепочке, она потеряет вознаграждение 50 CT, если потерпит неудачу.

Ключевым в этом примере здесь является требование кафе получить шесть подтверждений. Чем больше подтверждений потребует кафе, тем труднее будет Лизе построить ветвь более сильную, чем ветвь честных майнеров. Ей понадобится сказочное везение.

Когда кафе получило шесть подтверждений, Лиза отставала на 2 блока. Ей нужно создавать новые блоки быстрее честных майнеров и опередить их на 1 блок. Ее шансы невелики. Чем больше отставание, тем меньше шансов, как показано в табл. 7.3.

ПОДТВЕРЖДЕНИЯ

Потребовав шесть подтверждений, можно быть уверенным, что никто не попытается провести против вас атаку двойного расходования. Но чем выше стоимость сделки, тем выше экономическая целесообразность такой атаки.

Таблица 7.3. Вероятность успеха атакующего с его точки зрения

Необходимо блоков, чтобы перегнать (z)	Вероятность q_z, что атакующий догонит честную цепочку, обладая q% хешрейта					
	1%	10%	18% (Том)	36% (Лиза)	45%	50%
1	0.010101	0.111111	0.219512	0.562500	0.818182	1.000000
2	0.000102	0.012346	0.048186	0.316406	0.669421	1.000000
3	1.0e-06	0.001372	**0.010577**	0.177979	0.547708	1.000000
4	1.0e-08	0.000152	0.002322	**0.100113**	0.448125	1.000000
5	1.1e-10	0.000017	0.000510	0.056314	0.366648	1.000000
6	1.1e-12	1.9e-06	0.000112	0.031676	0.299985	1.000000
10	1.1e-20	2.9e-10	2.6e-07	0.003171	0.134431	1.000000

Вероятность q_z вычисляется по следующей формуле, где:

q = хешрейт атакующего; p = хешрейт честной ветви; z = отставание в блоках.

$$q_z = \begin{cases} 1 & \text{если } p \le q \\ \left(\frac{q}{p}\right)^z & \text{если } q > p \end{cases}$$

Посмотрите на колонку, соответствующую хешрейту 36% Лизы. При отставании в 3 блока она должна произвести на 4 блока больше, чем честные майнеры. Вероятность успеха атаки двойного расходования в этом случае оценивается как 0,10 — если она готова пытаться бесконечно. Скорее всего, она будет продолжать пытаться вечно, что несколько снижает вероятность успеха.

Том тоже попытался провести атаку двойного расходования

Представьте, что вместо Лизы атаку двойного расходования решил провести Том (рис. 7.28). Его хешрейт в два раза меньше, чем хешрейт Лизы, и составляет 100 Мхеш/с.

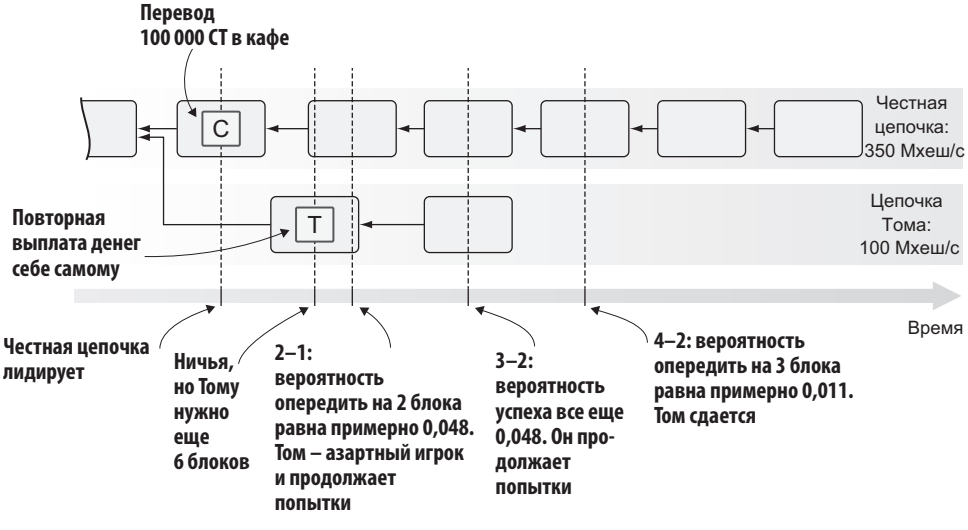


Рис. 7.28. Том пытается провести атаку двойного расходования с хешрейтом 18% и сдается. Ему повезло найти 2 блока примерно за то же время, в течение которого честные майнеры нашли 3

Том имеет еще меньше шансов, чем Лиза. Ему немного повезло вначале, и он быстро нашел 2 блока, но, отстав от честных майнеров на 2 блока, понимает, что его шансы слишком малы, и сдается. Понимание, что вероятность произвести на 3 блока больше, чем честные майнеры, равна примерно 0,011 ($z = 3$) — удручает.

Том — умный парень и понимает, что не стоит пытаться делать этого, что гораздо лучше защищать блокчейн вместе со всеми и получать свою справедливую долю вознаграждений, чем тратить время и силы на мошенничество. В конце концов, с 18% хешрейта он получает почти пятую часть всех вознаграждений за блок. Это более 50 CT в час. Те же самые 100 000 CT он сможет получить через 2000 часов, или 12 недель, никого не обкрадывая.

Том и Лиза договариваются совместно провести атаку двойного расходования

Том и Лиза вместе обладают суммарным хешрейтом 300 Мхеш/с. Они контролируют более половины (54,5%) общего хешрейта (рис. 7.29).

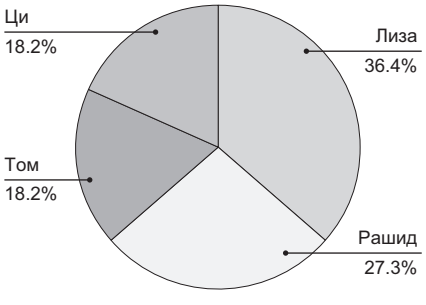


Рис. 7.29. Распределение хешрейта. Два майнера могут сговориться и контролировать большую часть хешрейта

Если они сговорятся совместными усилиями провести атаку двойного расходования и будут готовы пытаться до бесконечности, их шансы на успех равны 100% (см. табл. 7.3). Если они решат продолжать попытки не дальше чем на 50 блоков, их шансы на успех все еще будут очень близки к 100%.

Этот означает, что Том и Лиза могут переписать историю по своему желанию. Их суммарный хешрейт превышает хешрейт всех честных майнеров. Они могут создать ветку, начиная с любого блока в истории блокчейна, до-

гнать честную цепочку и превзойти ее. После этого все майнеры переключатся на ветвь Тома и Лизы. Обратите внимание, что они по-прежнему не могут украсть чьи-либо деньги в блокчейне, но могут заработать на мошенничестве с двойным расходованием столько, сколько захотят.

Давайте подумаем, что случится, если Том и Лиза начнут проводить атаки двойного расходования. Например, они могут купить кафе и добавить транзакцию, повторно расходующую те же средства, чтобы в итоге получить и кафе, и 100 000 СТ. Люди будут замечать, что история блокчейна периодически меняется. Раньше было достаточно шести подтверждений транзакций, но теперь этого мало. Что произойдет с ценностью жетонов на булочки, если надежность блокчейна уменьшится? И что произойдет с ценностью жетонов на булочки, когда люди услышат о продолжающихся атаках двойного расходования?

Возникнет паника! Люди не захотят связываться с этой ненадежной и небезопасной системой. Многие поспешат продать все свои жетоны на рынке жетонов вне кафе. Проблема в том, что покупателей не так много. Что случится с ценой жетонов в долларах при снижении спроса и увеличении предложения? Цена рухнет.

А что случится, когда цена рухнет? Паника усилится! Число людей, желающих продать жетоны, увеличится, что еще больше обрушит цены.

Майнинг начинает приносить все меньше прибыли Тому, Лизе и другим майнерам. Наконец, ценность вознаграждения за блок становится настолько низкой, что денег после продажи добытых жетонов не хватает на оплату счетов за электроэнергию. Им приходится покончить с майнингом, потому что это занятие начинает приносить убыток.

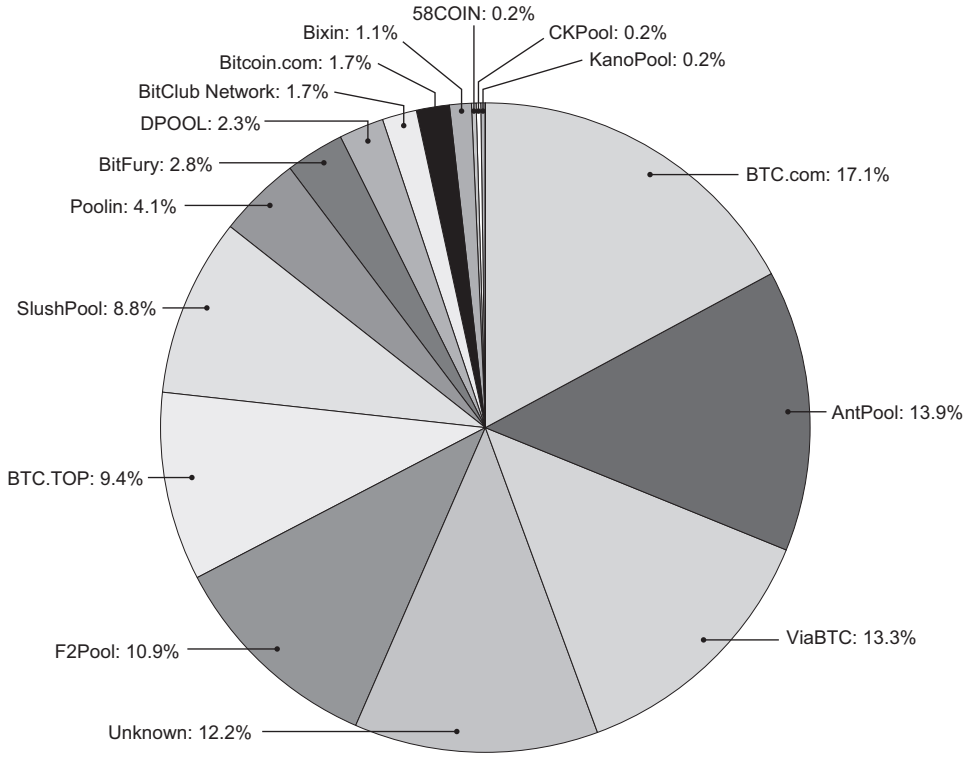
Том и Лиза должны дважды подумать, прежде чем начать атаковать систему, даже если у них есть все шансы на успех. Одного лишь факта, что два майнера контролируют более 50% общего хешрейта, может оказаться достаточно, чтобы вызвать снижение цен, потому что люди будут нервничать по поводу *централизации майнинга* — когда несколько человек контролируют большую часть общего хешрейта. Им даже не нужно атаковать систему, чтобы снизить ценность жетонов на булочки.

Ограничение централизации майнинга

Что могут предпринять люди, чтобы ограничить власть Тома и Лизы? Они сами могут начать заниматься майнингом. Допустим, в майнинг включились

РАСПРЕДЕЛЕНИЕ ХЕШРЕЙТА В СИСТЕМЕ БИТКОИН

На момент написания этой книги хешрейт системы Биткоин в 50 эксахешей в секунду распределялся следующим образом (источник: blockchain.info):



Распределение постоянно меняется, но эта диаграмма позволяет получить представление о том, как может выглядеть распределение в реальном мире.

еще пять человек и каждый добавил компьютер с вычислительной мощностью 150 Мхеш/с. Теперь у нас совершенно иная ситуация (рис. 7.30).

Общий хешрейт увеличился с 550 Мхеш/с до 1300 Мхеш/с. Крупнейший майнер, Лиза, с ее 200 Мхеш/с имеет только около 15% от общего хешрейта. Теперь уже не меньше пяти майнеров должны вступить в сговор, чтобы контролировать больше половины суммарного хешрейта, потому что четыре самых крупных майнера контролируют 49,9%.

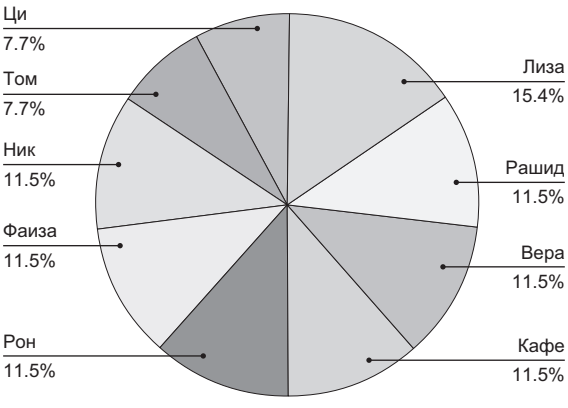


Рис. 7.30. Новое распределение хешрейта. Теперь намного сложнее получить контроль над преобладающей долей хешрейта

Люди имеют серьезные причины заняться майнингом: они владеют жетонами на булочки и хотят, чтобы система надежно защищала их деньги от панических падений цен из-за централизации майнинга. Обратите внимание: с увеличением числа майнеров уменьшается сумма вознаграждения, которое получает каждый из них. В какой-то момент некоторые майнеры — возможно, наиболее неэффективные — обнаружат, что майнинг не окупает расходуемой электроэнергии, и выключат свои майнинговые компьютеры. Рынок будет постепенно вытеснять неэффективных майнеров.

Комиссионные отчисления за транзакции

В настоящее время система имеет несколько майнеров, которые производят блоки независимо друг от друга. Это в значительной мере препятствует цензуре. Чтобы помешать транзакции оказаться в блокчейне, необходим сговор всех майнеров. Один майнер или часть майнеров могут только отложить подтверждение транзакции, но рано или поздно один из майнеров, не участвующих в сговоре, найдет действительное доказательство работы для блока с транзакцией и опубликует блок.

Все прекрасно. Но есть две проблемы:

- ✶ большие блоки обрабатываются медленнее;
- ✶ размер блока ограничен.

Эти два свойства оказывают некоторое влияние на выбор транзакций майнером. Начнем с первой из проблем, а затем обсудим влияние ограничения размера блока.

Большие блоки обрабатываются медленнее

Предположим, что Лиза и Том одновременно нашли действительные доказательства работы для своих блоков. Блок Лизы занимает 200 Кбайт и содержит 400 транзакций, а блок Тома — 100 Кбайт и 200 транзакций. Оба хотят, чтобы их блоки стали частью самой сильной цепи, но только один может занять это место. Они начинают выгружать свои блоки в общую папку одновременно (рис. 7.31).

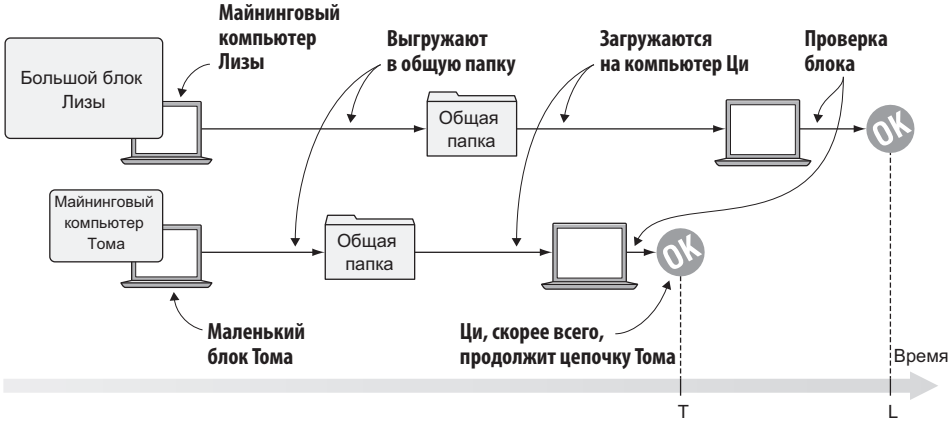


Рис. 7.31. Лиза и Том соревнуются за то, чтобы Ци и другие майнеры продолжили их цепочки. Том побеждает в этой гонке, потому что его блок меньше

Блок Тома меньше блока Лизы, а значит, Том выгрузит свой блок в общую папку раньше Лизы. Ци тоже раньше загрузит блок Тома. Наконец, Ци должна проверить загруженные блоки, прежде чем выбрать, за каким из них последовать. Меньший блок обычно проверяется быстрее, поэтому блок Тома будет проверен быстрее, чем блок Лизы.

В результате Ци выберет блок Тома как лучший текущий конечный блок в цепочке на момент времени T и продолжит цепочку вслед за блоком Тома. В момент времени T для Ци блока Лизы не существует, потому что Ци еще не проверила его. Она все еще продолжает загружать блок Лизы из общей

папки. Когда Ци наконец проверит блок Лизы в момент L, она уже выбрала блок Тома, а блок Лизы будет сохранен на случай будущих реорганизаций цепочки.

У майнеров есть явный стимул создавать маленькие блоки. С каждой дополнительной транзакцией, добавляемой в блок, они теряют немного конкурентоспособности в гонке блоков.

А причем тут комиссионные отчисления?

Вот тут и вступает в игру комиссия за транзакции. Если майнеру немного доплатить за каждую транзакцию, добавляемую в блок, это компенсирует потерю конкурентоспособности.

Люди, осуществляющие платежи, заинтересованы в скорейшем подтверждении своих транзакций в блокчейне. Разве не было бы здорово, если бы Джон мог зарезервировать немного денег в своей транзакции для майнера, который включит ее в ближайший блок? При таком подходе плательщик может компенсировать майнеру потерю конкурентоспособности.

Немного изменив обработку транзакций, можно организовать такую возможность. Представьте, что Джон хочет побыстрее купить булочку и, чтобы побудить майнеров включить его транзакцию в блок, он решает добавить комиссию за транзакцию. Он строит транзакцию, как показано на рис. 7.32.

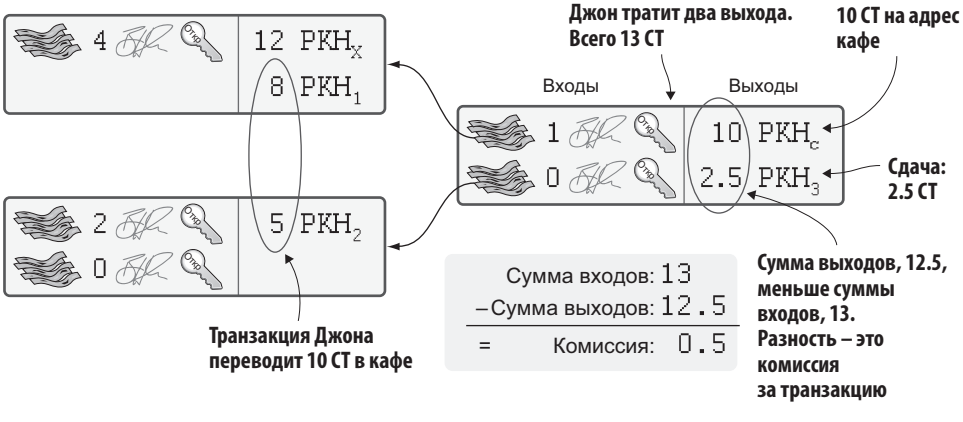


Рис. 7.32. Джон включает комиссию для майнера, который создаст блок с этой транзакцией

Когда Джон создавал аналогичную транзакцию в главе 5, сумма ее входов была равна сумме выходов. Он не платил комиссию за транзакцию.

На этот раз Джон хочет добавить небольшую комиссию за транзакцию. Он тратит два входа, всего 13 СТ, и добавляет выход 10 СТ для кафе и выход со сдачей 2,5 СТ для себя. Затем подписывает транзакцию, как обычно, и отправляет ее всем майнерам.

Лиза, будучи майнером, получает эту транзакцию и замечает, что в ней есть комиссия в 0,5 СТ. Она хочет получить эту комиссию, решив, что она в достаточной мере компенсирует небольшой дополнительный риск проигрыша в гонке за блок из-за включения транзакции.

ПОЛОВИНА СТ?

Жетоны на булочки и биткоины можно делить на более мелкие части. Наименьшая единица в Биткоин — *сатоши*:
1 сат = 10^{-8} биткоина.

Джон может настроить вознаграждение майнерам за включение его транзакции. Если ему важно получить подтверждение транзакции в одном из следующих нескольких блоков, он должен заплатить относительно высокую комиссию. Если скорость не важна, он может заплатить меньшую комиссию, но ему нужно быть осторожным. Если он заплатит слишком маленькую комиссию, ни один майнер не будет заинтересован побыстрее подтвердить его транзакцию.

Подробнее о комиссии за транзакцию и как ее изменить, если транзакция застрянет в ожидании подтверждения (эта операция известна как *повышение комиссионных*), мы поговорим в главе 9.

Когда Лиза решает, включать транзакцию в блок или нет, для нее имеют значение только размер транзакции и сумма комиссии за нее. Фактически ее интересует *плата за байт*. Транзакция Джона имеет размер около 400 байт и содержит комиссию 0,5 СТ, то есть 0,00125 СТ/байт. Эти простые вычисления Лиза выполняет для всех транзакций. Если плата за байт превышает определенный порог, она включит транзакцию в блок.

Она может выбирать транзакции по своему усмотрению, как описано в разделе

КОМИССИЯ В БИТКОИН

На момент написания книги комиссии в 4 сат/байт было вполне достаточно, чтобы обеспечить включение транзакции в один из следующих шести блоков. Учитывая, что типичная транзакция имеет размер 500 байт, комиссия за нее составит 0,00002 BTC, или около 20 центов.

«Выбор транзакций» в главе 6. Например, она может включить свою собственную транзакцию без какой-либо комиссии или отбросить все транзакции, выполняющие оплату булочек, независимо от размера комиссии. И это нормально. Разные майнеры могут иметь разные стратегии выбора транзакций. Большинство, вероятно, будут принимать решения, основываясь только на плате за байт.

Как Лиза получает эти комиссионные? Используя свою монетарную транзакцию (рис. 7.33).

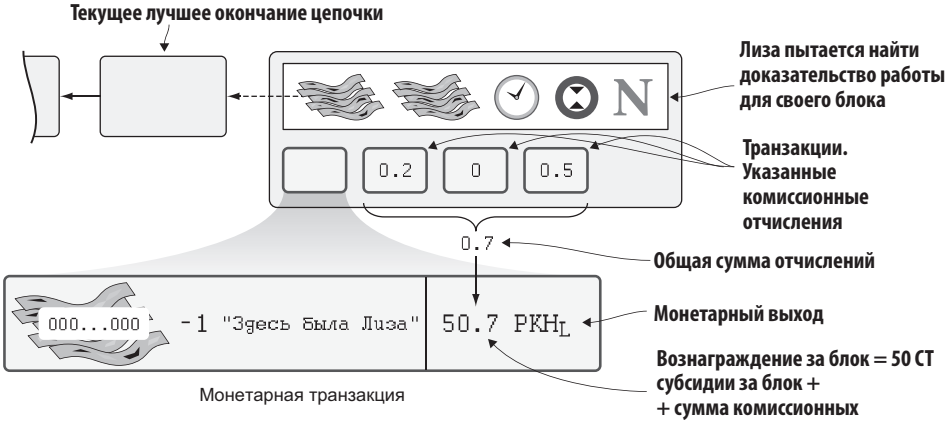


Рис. 7.33. Лиза приступает к созданию блока и включает в него транзакцию Джона и еще несколько других транзакций. Она суммирует комиссионные отчисления в выходе своей монетарной транзакции

Лиза суммирует все комиссионные сборы из транзакций в своем блоке и увеличивает выход монетарной транзакции на эту сумму. То есть выход монетарной транзакции — вознаграждение за блок — это субсидия за блок (50 CT) плюс сумма всех комиссионных сборов из транзакций в блоке. Обратите внимание, что мы расширили термин *вознаграждение за блок*, включив *субсидию на блок* (вновь созданные деньги) и комиссию за транзакцию.

Подготовив блок, Лиза начинает поиск действительного доказательства работы для него.

Размер блока имеет ограничения

Блоки не могут увеличиваться в размерах до бесконечности. Максимальный размер составляет 1 000 000 байт, но подробнее этот вопрос мы обсудим

в разделе «Ограничение размера блока» в главе 10. Если подтверждения ожидает больше транзакций, чем может уместиться в блоке, майнеры должны выбрать, какие транзакции включить в блок, а какие исключить.

Комиссия за транзакцию играет важную роль в этой ситуации, потому что более высокая комиссия стимулирует майнеров включить транзакцию в блок в первую очередь. Комиссия не только компенсирует снижение конкурентоспособности майнера, но также помогает конкурировать с другими транзакциями за место в блоке. Эта ситуация называется *рынком комиссий* (рис. 7.34).

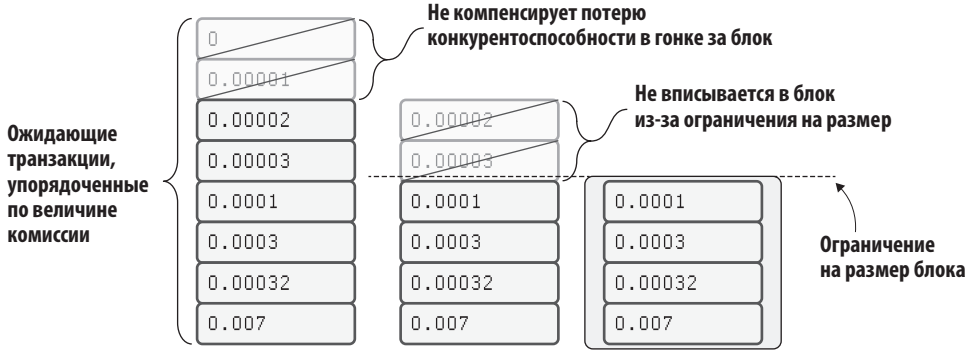


Рис. 7.34. На рынке комиссий транзакции конкурируют за место в блоке. Числа в транзакциях обозначают величину комиссии в СТ/байт

Если доступно больше блоков, чем транзакций, ожидающих подтверждения, конкуренция между транзакциями отсутствует (рис. 7.35).

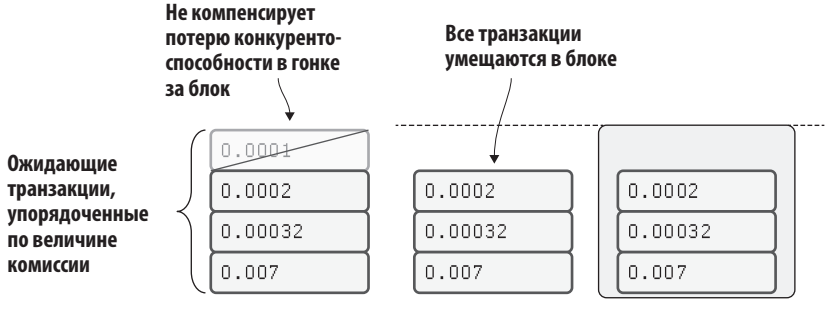


Рис. 7.35. В отсутствие рынка комиссий транзакции не конкурируют между собой. Они просто компенсируют потерю конкурентоспособности

В этой ситуации будет подтверждена любая транзакция, несущая компенсацию за потерю конкурентоспособности.

На момент написания этой книги комиссионные отчисления возрастали время от времени, в периоды всплеска интереса к Биткоину. Но иногда, когда ожидающих транзакций оказывалось немного, комиссия опускалась до минимального уровня и составляла 1 сат/байт, то есть 0,000 000 01 BTC/байт.

Когда субсидия за блок станет равной 0

Как уже говорилось в главе 2, субсидия за блок будет уменьшаться вдвое примерно каждые четыре года. В какой-то момент субсидия окажется не настолько большой, чтобы стимулировать майнеров. Если величина вознаграждения за блок опустится ниже стоимости потраченной электроэнергии, какой смысл в майнинге?

Когда наступит это время, преобладающее значение для майнеров будут иметь комиссионные отчисления за транзакции. Типичный майнер хочет, чтобы доход от майнинга как минимум покрывал счета за электроэнергию (рис. 7.36).

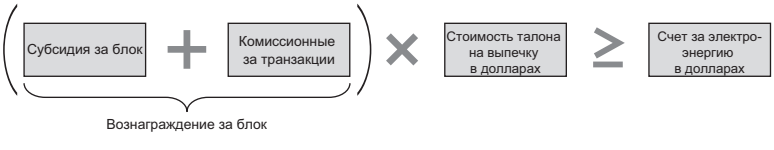


Рис. 7.36. Майнер должен получать достаточно денег, чтобы оплачивать счета за электроэнергию

Обратите внимание, что *стоимость* субсидии за блок не обязательно должна уменьшаться с течением времени. Некоторые примеры перечислены в табл. 7.4.

Таблица 7.4. В абсолютном выражении субсидия за блок может уменьшаться вдвое, а в денежном выражении — зависеть от стоимости жетона на булочки

Субсидия за блок	Цена 1 CT	Цена субсидии за блок
50 CT	$0,10	$5
25 CT	$0,25	$6,25

Как видите, абсолютное выражение субсидии за блок не является мерой дохода от майнинга. Мы должны учесть *стоимость* субсидии и *стоимость* комиссионных за транзакции. Одно можно сказать наверняка: когда величина субсидии опустится до нуля, ее стоимость также опустится до нуля. В какой-то момент размер субсидии за блок перестанет быть достаточным стимулом для майнинга.

LIGHTNING NETWORK

Дополнительную информацию о Lightning Network можно найти на веб-ресурсе 16 (приложение B). К сожалению, в книге не так много места, чтобы в полной мере осветить эту сложную и интересную тему.

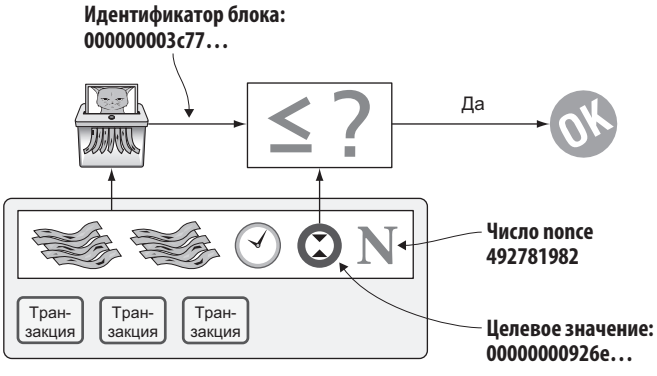
Когда это произойдет, комиссионные за транзакции станут основным доходом для эффективных майнеров. Если Джон захочет, чтобы его транзакция была подтверждена, он должен будет заплатить достаточно большую комиссию, чтобы один или несколько майнеров заинтересовались включением этой транзакции в блок. Так выглядит конкуренция за место в блоке.

Мы можем только догадываться, как вырастет размер комиссионных в будущем. Некоторые утверждают, что комиссия в Биткоин уже слишком высока для типичных сценариев использования. По мере увеличения комиссионных за транзакции, для таких случаев, как платежи с небольшими суммами, придется найти другие способы обработки. На основе Биткоин разрабатываются новые системы, которые позволяют людям объединить практически бесконечное количество платежей в одну или две транзакции. Особый интерес вызывает одна из таких систем: Lightning Network. Если в одной транзакции Биткоин можно будет сделать миллион платежей, все эти платежи смогут разделить стоимость комиссии за транзакцию.

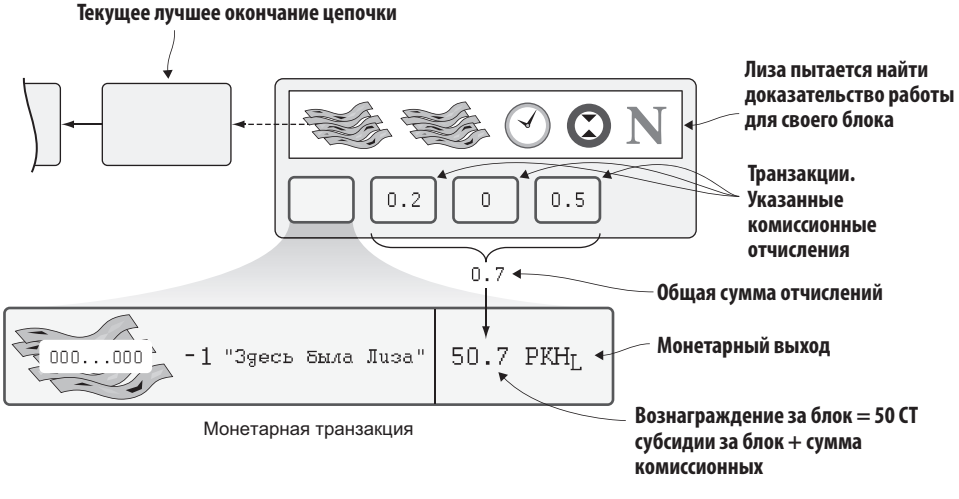
Повторение

В этой главе мы решили проблему цензуры транзакций. Лиза могла единолично решать, какие транзакции включать в блокчейн. Мы устранили эту проблему, добавив возможность привлечь к работе много «Лиз», или майнеров. Теперь кошельки могут отправлять свои транзакции любым или всем майнерам и надеяться, что кто-то из них обработает их.

Майнеры соревнуются за право добавить следующий блок в блокчейн. Такое право получает первый нашедший действительное доказательство работы для своего блока.

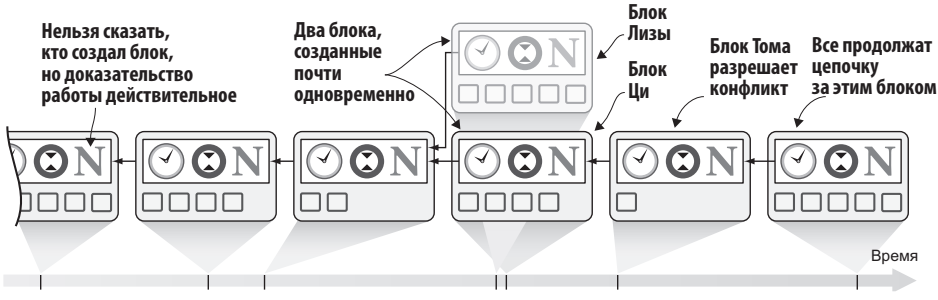


Майнер, выигравший гонку, опубликует свой блок и получит вознаграждение, состоящее из субсидии за блок и комиссий за транзакции. Вознаграждение аккумулируется в монетарной транзакции.



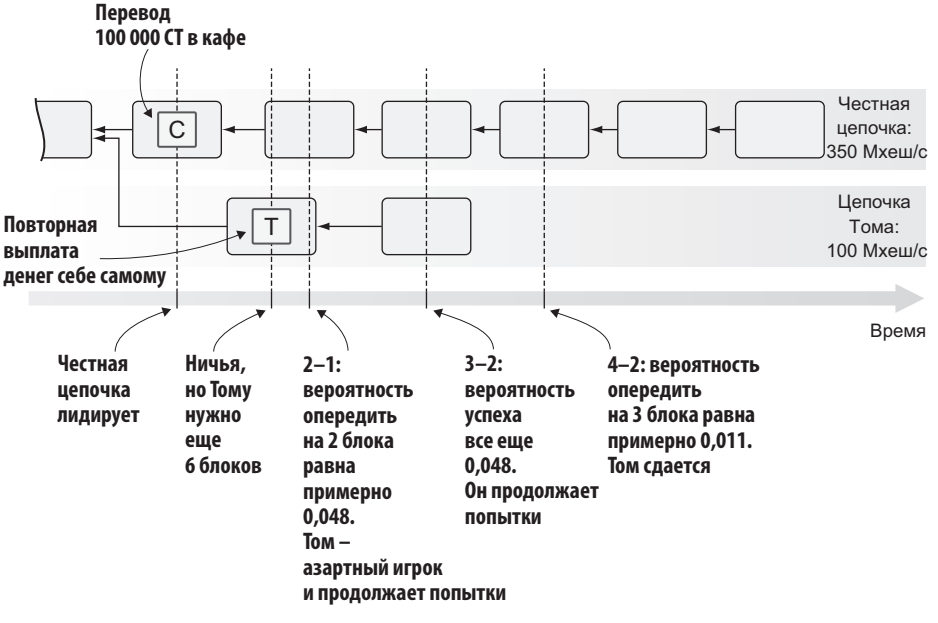
Субсидия за блок обеспечивает честную эмиссию денег до того момента, пока не будут отчеканены все 21 000 000 новых жетонов на булочки. Отправитель транзакции добавляет комиссию за транзакцию, чтобы стимулировать майнеров ко включению его транзакции в свои блоки.

Это состязание приводит к естественному расщеплению, если два майнера найдут блок примерно в одно и то же время. Рано или поздно эти конфликты будут разрешены.



Решение зависит от того, какую ветвь выберут другие майнеры. Обычно майнеры выбирают первый увиденный ими действительный блок.

Продавец не должен доверять транзакции с высокой стоимостью, пока за блоком с транзакцией не будет создано достаточно большое количество других блоков. Это снижает риск двойного расходования средств.



Попытка выполнить атаку двойного расходования средств может дорого стоить майнеру. Если атака не удастся, майнер потратит много электроэнергии и потеряет вознаграждение за блок. Количество необходимых подтверждений выбирается продавцом и должно учитывать стоимость транзакции.

Изменения в системе

Доказательство работы заменило подписи блоков, введенные в главе 6, и мы можем удалить их из таблицы понятий (табл. 7.5).

Таблица 7.5. Понятие доказательства работы из Биткоин заменило подписи блоков. Лиза стала одним из нескольких майнеров

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
~~Лиза~~	~~Майнер~~	~~Глава 7~~
~~Подпись блока~~	~~Доказательство работы~~	~~Глава 7~~
Общая папка	Сеть Биткоин	Глава 8

Сейчас Лиза решает те же задачи, что и майнеры в системе Биткоин, поэтому уберем из таблицы и Лизу. Общая папка — последнее, что осталось от системы жетонов на булочки и о чем нам предстоит позаботиться в следующей главе.

Пришло время выпустить новую версию системы жетонов на булочки (табл. 7.6).

Таблица 7.6. Примечания к релизу, жетоны на булочки 7.0

Версия	Особенность	Как реализована
НОВАЯ 7.0	Противодействие цензуре	В работу по поиску доказательства работы включается несколько майнеров
	Любой может включиться в майнинг	Автоматическое регулирование сложности
6.0	Лиза лишена возможности удалять транзакции	Подписи блоков в блокчейне
	Полная проверка узлами	Узлы загружают и проверяют весь блокчейн
	Легкие кошельки экономят трафик	Фильтры Блума и доказательство Меркла
5.0	Расходование нескольких «монет» в одном платеже	Несколько входов в транзакции
	Любой может проверить электронную таблицу	Подписи в транзакциях доступны всем
	Отправитель может определять критерии расходования денег	Программы на языке Script внутри транзакции

Упражнения

Для разминки

7.1. Каким образом Лиза осуществляла единоличную власть в главе 6?

7.2. Почему возможность цензуры транзакций уменьшается с появлением нескольких майнеров?

7.3. Методика на основе выпадения счастливых чисел давала неплохие результаты. Почему мы отказались от нее? Почему эта идея оказалась непригодной?

7.4. Как проверить действительность доказательства работы?

7.5. Как майнер генерирует действительное доказательства работы?

7.6. Что подразумевается под словами *сильнейшая цепочка*?

7.7. Что означает фраза: «Майнер имеет хешрейт 100 Мхеш/с»?

7.8. Период корректировки только что закончился, на создание последних 2016 блоков ушло 15 дней. Как изменится целевое значение — увеличится или уменьшится?

7.9. С каким процентом хешрейта можно надеяться на успех атаки двойного расходования при условии, что вы готовы повторять попытки до бесконечности?

Придется пораскинуть мозгами

7.10. Допустим, что одновременно были созданы большой и маленький блоки. Почему большой блок имеет меньше шансов стать частью сильнейшей цепочки, чем маленький?

7.11. Предположим, что в середине периода корректировки скорость создания блоков неожиданно увеличивается вдвое, с 6 блоков в час до 12. Ничего другого экстраординарного в течение периода ретаргетинга не происходило. Что произойдет с целевым значением по окончании этого периода?

7.12. Допустим, Селма обладает 52% общего хешрейта. Она решает уменьшить период корректировки в своем программном обеспечении с 2016 блоков (две недели) до 144 блоков (один день). Все остальные не приняли ее идею и продолжают использовать старое программное обеспечение. Что

случится после окончания периода корректировки у нее, когда она скорректирует свое целевое значение? Примут ли остальные майнеры и полные узлы блоки, созданные Селмой? Кто пострадает в этой ситуации?

7.13. Почему майнер может отказаться подтверждать транзакцию с очень маленькой комиссией?

Итоги

- Наличие нескольких майнеров позволяет избежать концентрации власти в одних руках и предотвратить цензуру транзакций.
- Доказательство работы используется, чтобы выбрать, кто сможет добавить следующий блок.
- Доказательство работы позволяет любому начать майнинг, не спрашивая разрешения у кого бы то ни было.
- Целевое значение автоматически корректируется через каждые 2016 блоков, чтобы сохранить эмиссию на установленном уровне.
- Комиссионные за транзакцию стимулируют майнеров ко включению транзакции в их блоки.
- Чтобы снизить риск двойных расходов, получатель жетонов на булочки или биткоинов выбирает обязательное количество подтверждений.
- Майнер получает за блок такое вознаграждение, какое заслуживает. Чем большей долей хешрейта в системе он обладает, тем большую долю вознаграждений получает.
- Чем сильнее цепочка, чем больше накопленных доказательств работы — тем труднее переписать эту цепочку.

8 Одноранговая сеть

Эта глава охватывает следующие темы:

- ✓ устранение последнего центрального органа — общей папки;
- ✓ передача транзакций в одноранговой сети;
- ✓ оставляем в прошлом жетоны на булочки;
- ✓ запуск одноранговой сети.

Теперь поговорим о слоне в комнате — острой проблеме, которую до сих пор мы обходили стороной, — общей папке. Все блоки, производимые майнерами, сначала попадают в общую папку, откуда загружаются другими полными узлами и майнерами. В этой главе мы наконец избавимся от центральной общей папки и заменим ее *децентрализованной одноранговой сетью* (рис. 8.1). Одноранговая сеть позволяет полным узлам (и майнерам) посылать блоки друг другу напрямую. Позволив узлам общаться друг с другом, мы сможем избавиться от центрального узла, используемого для связи.

Еще одна проблема, о которой мы мало говорили, — как кошельки отправляют транзакции майнерам по электронной почте. Когда новый майнер включается в работу, все кошельки должны обновить свои списки майнеров. Не самое лучшее решение. А благодаря одноранговой сети узлов кошельки смогут рассылать свои транзакции всем майнерам, не зная, кто они и где находятся.

Мы проследим весь путь транзакции через сеть, от создания неподтвержденной транзакции до включения ее в блок. Путь транзакции начинается

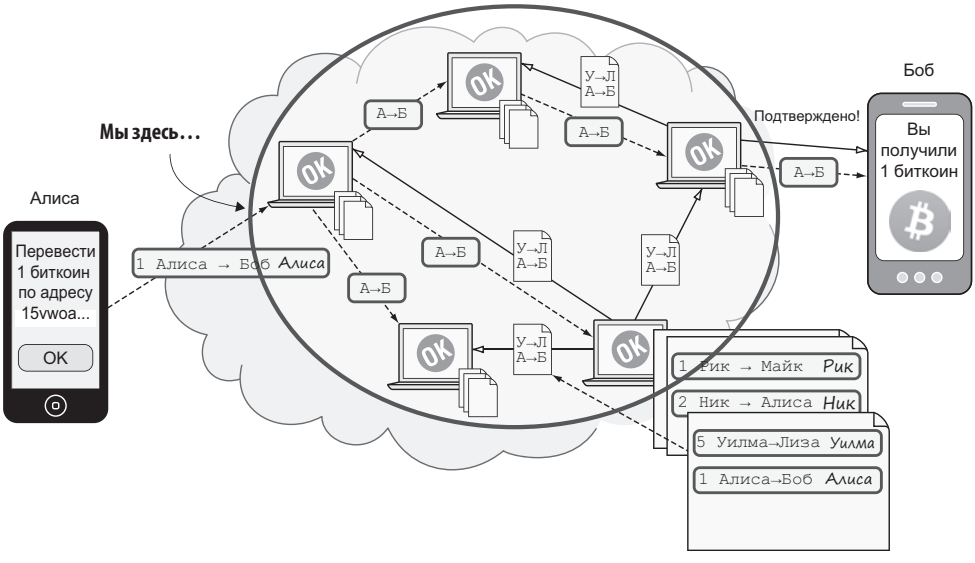


Рис. 8.1. Одноранговая сеть Биткоин

в кошельке Джона и заканчивается в блокчейне, после чего посылается уведомление в кошелек Боба.

Проследив движение транзакции через систему, мы оставим в прошлом жетоны на булочки, которые использовали, чтобы понять суть работы системы Биткоин. С этого момента мы будем говорить только о Биткоин, потому что практически все различия между булочками и Биткоин исчезнут. Такие упрощения потеряют смысл — ведь на самом деле мы хотим больше узнать о Биткоин!

В последнем разделе этой главы вы узнаете, как новый узел подключается к одноранговой сети и становится ее частью. Это далеко не тривиально. Как он находит узлы для подключения? Как загружает блокчейн до последнего блока? Мы разберемся со всеми этими вопросами. В конце главы вы узнаете, как настроить собственный полный узел.

Общая папка

Люк, администратор общей папки, играет роль центрального органа (рис. 8.2). В конечном итоге он решает, какие блоки можно хранить в общей папке. Он также решает, кто может читать и писать в общую папку.

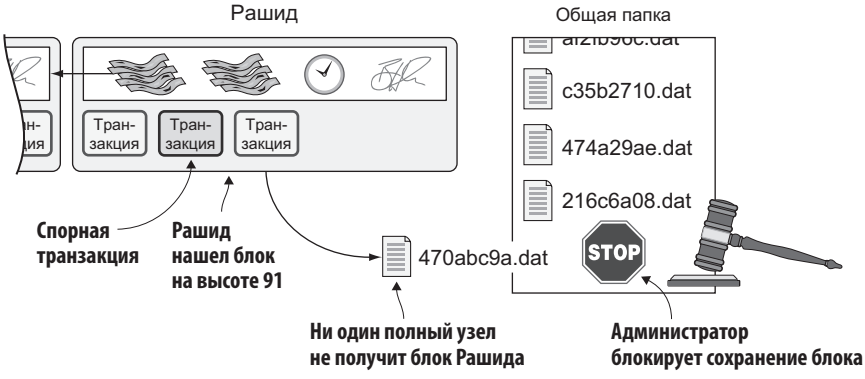


Рис. 8.2. Общая папка играет роль центрального органа

До сих пор мы считали Люка абсолютно нейтральным и честным парнем, но что, если мы ошибались или сотрудники Acme Insurance заставили его удалять определенные блоки? Какой смысл доказывать работу, если система может подвергаться цензуре на уровне блоков? Доказательство работы обеспечило невозможность цензуры *транзакций*, потому что пользователи получили возможность отправлять транзакции нескольким майнерам. Но *блоки*, содержащие транзакции, все еще могут удаляться любым, кто имеет доступ к общей папке, с привилегиями администратора. Проще говоря, система все еще не защищена от цензуры. Пока кто-то единолично может решать, какие блоки или транзакции имеют право на существование, система не защищена от цензуры.

Общая папка порождает еще одну проблему. Представьте, что Рашид создал блок размером 1 Мбайт и опубликовал его в общей папке. Все, кто просматривает общую папку, то есть все полные узлы, одновременно начнут загрузку блока Рашида. Если в сети работает 100 полных узлов, общий объем данных, который необходимо отправить из общей папки разным узлам, составит 100 Мбайт. Это может значительно замедлить *распространение блока* — передачу его от создателя всем остальным узлам. Чем больше узлов, тем медленнее будут распространяться блоки.

Создание одноранговой сети

А что, если полные узлы и майнеры смогут напрямую общаться друг с другом, не полагаясь на центральную общую папку? В этом случае они могли бы посылать блоки друг другу через одноранговую сеть (рис. 8.3).

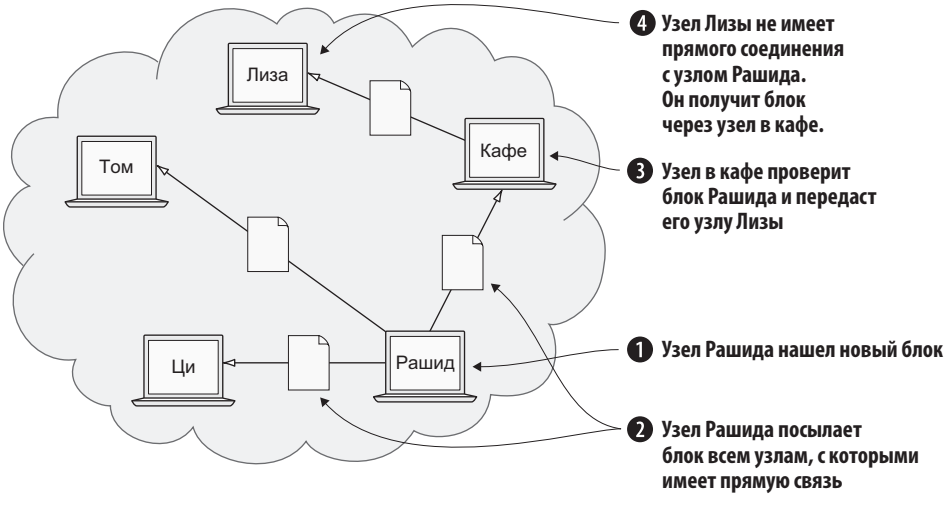


Рис. 8.3. В одноранговой сети блоки передаются от одного узла к другому, подобно слухам между людьми

Одноранговую сеть можно представить как толпу людей. Никто не знает всех остальных в толпе, но может быть знаком с тремя из них. Когда происходит что-то интересное — например, Рашид находит блок, — он рассказывает об этом своим трем друзьям, которые, в свою очередь, рассказывают эту новость своим друзьям, и так далее, пока все не узнают о новом блоке. Мы называем такие сети *сетями распространения слухов* (gossip networks).

Теперь остановить распространение блоков будет очень непросто. Узел может отказаться принимать или пересылать блок соседним узлам, но эти соседние узлы подключены к нескольким другим узлам, которые с радостью передадут им блок. Один узел практически ничего не сможет сделать для цензуры информации.

ПЕРЕСЫЛКА

Под пересылкой полученного блока подразумевается передача его другим узлам.

Предположим, что Рашид нашел блок и хочет передать его всем узлам. Он посылает свой блок узлам Ци, Тома и в кафе. По какой-то причине узел в кафе не пересылает блок Лизе (рис. 8.4). Но у Лизы есть несколько соседей в этой сети. Она связана с Томом и Ци. Том сообщит Лизе о новом блоке и отправит его ей. Кафе не сможет скрыть информацию от Лизы, пока у нее есть хорошие связи, то есть много разных соседей в сети.

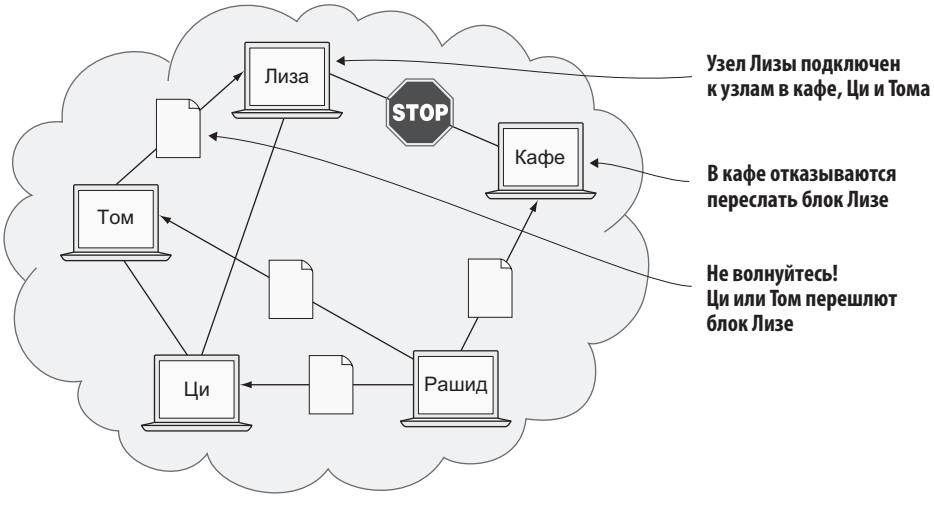


Рис. 8.4. Если в кафе откажутся переслать блок Лизе, это сделает кто-то другой

Теперь, когда у нас есть эта замечательная сеть, кошельки могут использовать ее для отправки транзакций майнерам. Теперь им не придется хранить списки электронных адресов майнеров. Транзакции будут распространяться по одноранговой сети и попадать на все полные узлы в течение нескольких секунд. В том числе и на узлы майнеров, которые тоже являются полными узлами. Об этом говорилось в главе 1 и повторяется на рис. 8.5.

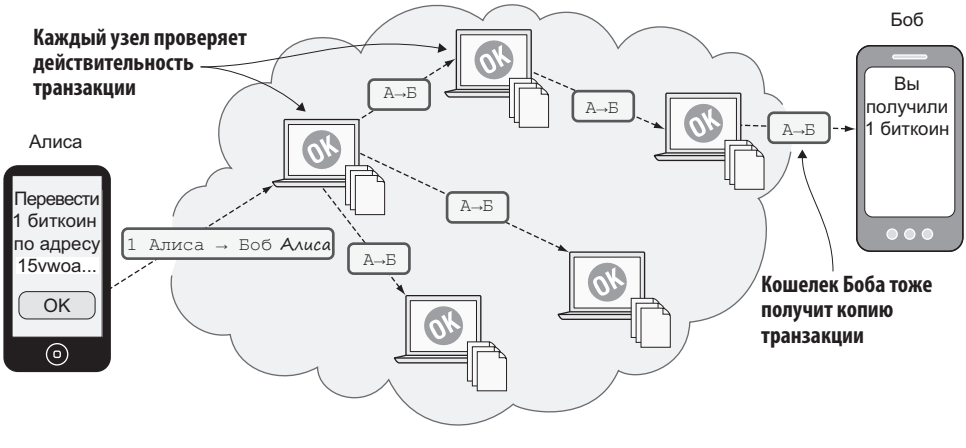


Рис. 8.5. Транзакции распространяются в одноранговой сети в точности как блоки. Кошелькам больше не нужно знать адреса майнеров

К транзакциям относится все то же, что и к блокам: один узел не может препятствовать их распространению по сети. Еще один замечательный эффект от использования одноранговой сети для распространения транзакций — получатель транзакции может быть уведомлен, что транзакция находится в состоянии *ожидания* и, возможно, вскоре будет подтверждена. Как это работает, мы увидим чуть позже.

Как общаются соседние узлы?

Давайте посмотрим, как происходит общение между двумя соседними узлами. В частности, посмотрим, как узел Тома соединяется с узлом Лизы и как они общаются через свой канал связи, который называется TCP-соединением (Transmission Control Protocol — протокол управления передачей) (рис. 8.6).

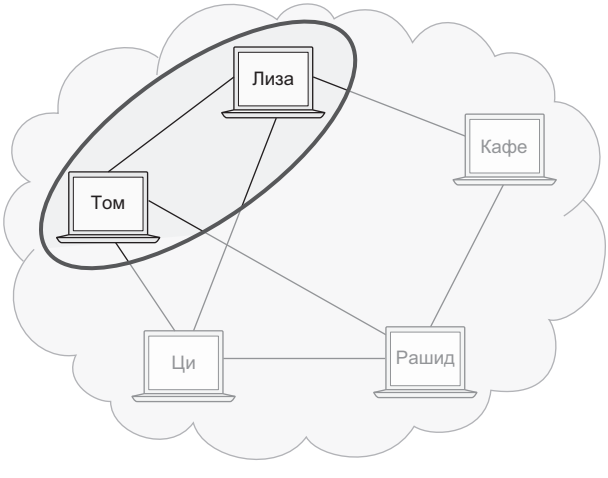


Рис. 8.6. Узлы Тома и Лизы общаются через Интернет, создавая канал связи между собой

TCP

Когда вы открываете веб-страницу https://bitcoin.org, веб-браузер устанавливает TCP-соединение с *bitcoin.org*, загружает веб-страницу через это соединение и отображает ее.

Предположим, что узел Тома знает о существовании узла Лизы. В разделе «Инициализация сети» я расскажу, как узел Тома узнает о существовании других узлов. А пока предположим, что он знает *IP-адрес* и *номер порта* узла Лизы. Теперь он хочет подключиться к узлу Лизы, чтобы обмениваться информацией с ним. Все компьютеры в Интернете имеют IP-адрес, по

которому один компьютер может отправлять информацию другому. Компьютерная программа, принимающая входящие запросы на соединение, должна прослушивать определенный порт на своем компьютере. Компьютер Лизы имеет IP-адрес 142.12.233.96 и выполняет программу, реализующую функции системы жетонов на булочки, которая прослушивает порт 8333 на наличие входящих запросов на соединение.

Узел Тома соединяется с узлом Лизы, используя IP-адрес 142.12.233.96 и TCP-порт 8333. Его узел (компьютерная программа) обращается к своей операционной системе (ОС) с просьбой установить соединение с узлом Лизы (рис. 8.7). ОС посылает запрос компьютеру Лизы, в котором сообщает, что узел Тома хочет установить связь с компьютерной программой на узле Лизы через порт 8333. Ее компьютер знает, что порт 8333 прослушивается выполняющейся на нем программой, поэтому отправляет ответ: «Конечно, добро пожаловать». Компьютер Тома подтверждает получение ответа, отправляя обратно: «Отлично. Давай поговорим...».

ПОРТ 8333

Порт 8333 — это порт по умолчанию, который прослушивается программным обеспечением Bitcoin Core, широко используемым на полных узлах.

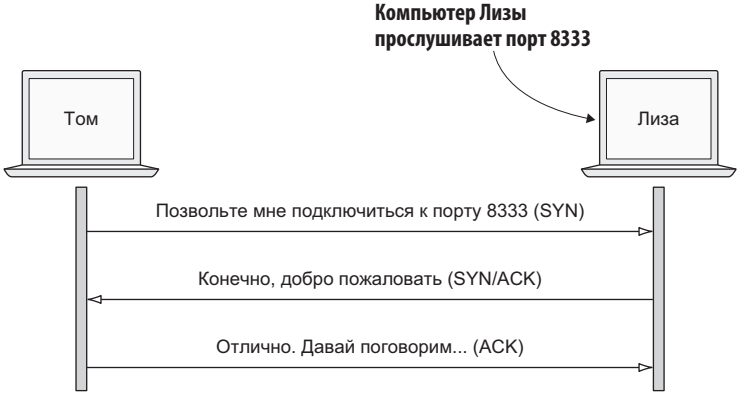


Рис. 8.7. Программа на компьютере Тома устанавливает TCP-соединение с компьютером Лизы. После этого они могут обмениваться данными

Программное обеспечение полного узла на компьютерах Тома и Лизы не участвует в этом обмене — соединение устанавливается их операционными системами, такими как Linux, Windows или macOS. По завершении обмена начальными сообщениями ОС передает готовое соединение программному

обеспечению узла. Теперь узлы Лизы и Тома могут свободно общаться друг с другом. Используя этот канал связи, или *TCP-соединение*, Том может отправлять данные Лизе, а Лиза — Тому.

Сетевой протокол

Том и Лиза теперь могут отправлять и получать данные по каналу связи. Но если узел Тома будет говорить на языке, который не понимает узел Лизы, общение потеряет всякий смысл (рис. 8.8). Узлы должны говорить на одном языке, то есть использовать общий *протокол*.

Рис. 8.8. Лиза должна понимать, что Том записывает в канал

ЭТО ВСЕГО ЛИШЬ АБСТРАКЦИЯ

В действительности сетевые сообщения выглядят не совсем так; я даю лишь абстрактное представление сообщений. Обсуждение точного формата сетевых сообщений выходит далеко за рамки этой книги.

Сетевой протокол в системе жетонов на булочки определяет набор поддерживаемых типов сообщений. Типичное сообщение в сети жетонов на булочки (то есть Биткоин) — это сообщение `inv` (рис. 8.9).

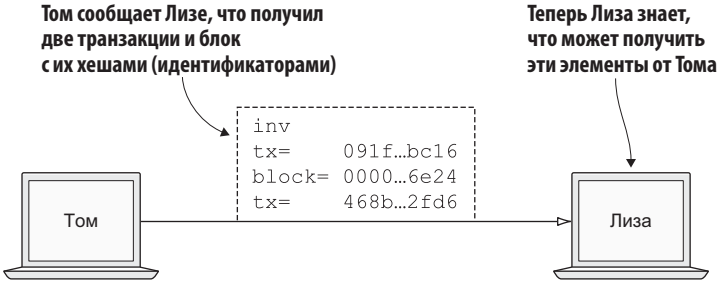


Рис. 8.9. Типичное сетевое сообщение

Узел использует сообщение `inv` — сокращенно от *inventory* (опись), — чтобы сообщить другим узлам, что у него есть. На рис. 8.9 узел Тома сообщает узлу Лизы, что может предложить две транзакции и блок. Сообщение содержит идентификаторы всех трех элементов.

Джон посылает транзакцию

Давайте проследим путь транзакции по сети от начала до конца и посмотрим, какие сетевые сообщения используются. Предположим, что одноранговая сеть уже инициализирована. К вопросу *инициализации* сети мы вернемся позже в этой главе.

В разделе «Легкие кошельки» в главе 6 говорилось, что кошельки могут подключаться к полным узлам и получать информацию обо всех заголовках блоков и транзакциях, интересующих их, используя фильтры Блума и доказательства Меркла (рис. 8.10).

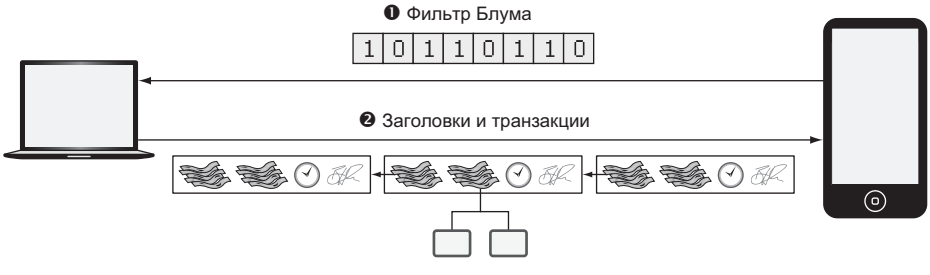


Рис. 8.10. Легкие кошельки взаимодействуют с узлами, используя сетевой протокол Биткоин

Не буду вдаваться в детали, как протекает это общение, но отмечу, что при этом используется тот же протокол, который используют узлы для общения друг с другом. Кошельки и полные узлы (включая майнеров) говорят на одном «языке».

Предположим, что Джон решил купить булочку в кафе. Кошелек Джона имеет установленное TCP-соединение с узлом Тома. Он сканирует платежный URI из кошелька кафе, после чего создает и подписывает транзакцию. Как это делается, вы уже знаете. Затем он отправляет транзакцию узлу Тома (рис. 8.11).

Вся эта процедура выполняется в три этапа. Кошелек Джона не просто отправляет транзакцию без спросу: он сначала сообщает узлу Тома, что у него есть транзакция, которую хотел бы отправить (рис. 8.12).

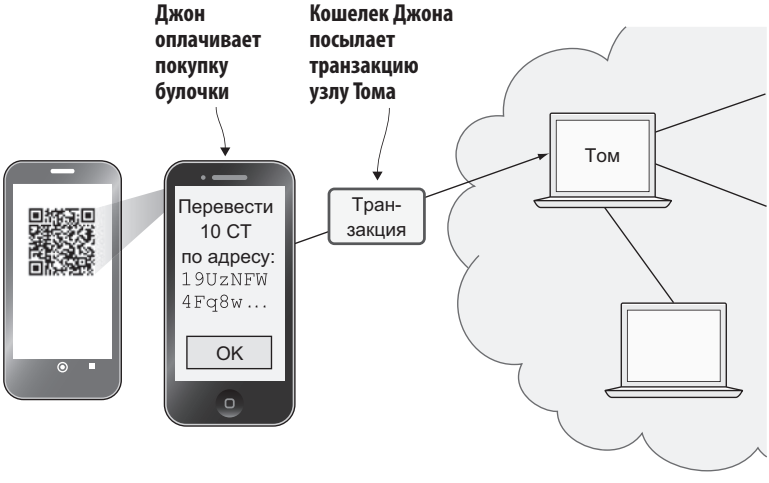


Рис. 8.11. Транзакция отправляется узлу Тома через TCP-соединение

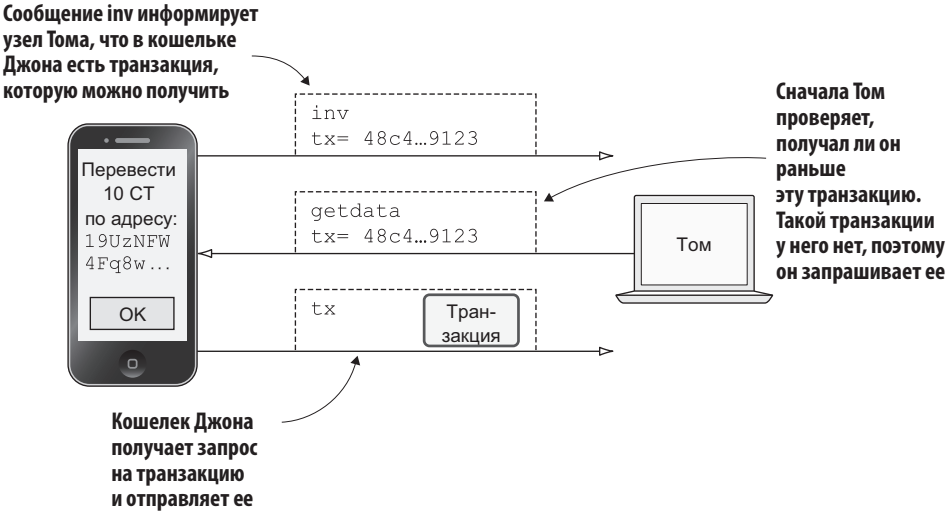


Рис. 8.12. Кошелек Джона сообщает узлу Тома, что у него есть транзакция, чтобы тот мог запросить ее

Первым посылается сообщение `inv`, как описано в предыдущем разделе. Кошелек Джона отправляет это сообщение полному узлу Тома. Том проверяет, не получал ли он раньше эту транзакцию. Конечно же, такая транзакция у него отсутствует, потому что кошелек Джона только что создал ее и еще никому не отправлял. Узел Тома решает получить эту транзакцию и посы-

лает сообщение-запрос `getdata`, которое выглядит как сообщение `inv`, но имеет другой смысл: `getdata` означает «я хочу получить это», тогда как `inv` означает «у меня есть что предложить».

Кошелек Джона получает сообщение `getdata` и посылает узлу Тома сообщение `tx`, содержащее транзакцию целиком. Том проверяет транзакцию и сохраняет ее, а затем пересылает эту транзакцию своим соседям в сети.

Вы можете спросить: «Почему кошелек Джона сразу не отправляет транзакцию целиком? К чему все эти реверансы с сообщениями `inv` и `getdata`?» Причины станут очевидны чуть позже, но если вкратце, то дело в том, что узлы могут уже иметь транзакцию; такой порядок действий, когда вместо целых транзакций отправляются только их хеши, помогает сэкономить сетевой трафик.

Том пересылает транзакцию

Если транзакция действительная, узел Тома проинформирует о ней своих соседей (рис. 8.13), используя сообщение `inv`, как это делал кошелек Джона, когда сообщал узлу Тома о транзакции.

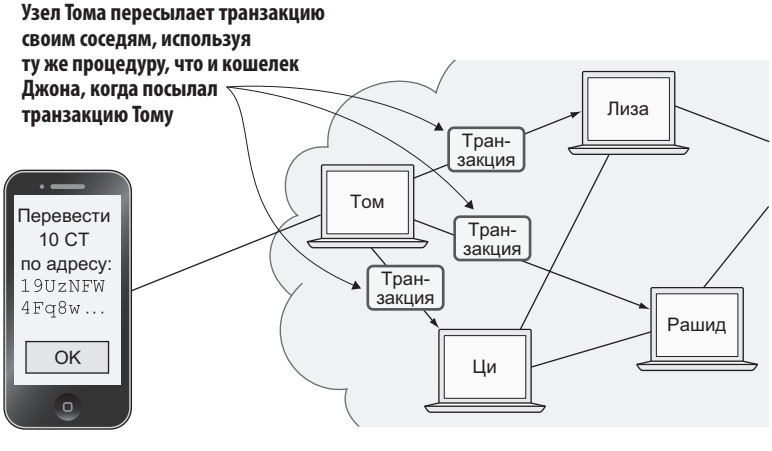


Рис. 8.13. Том пересылает транзакцию своим соседям

При этом выполняется тот же процесс обмена тремя сообщениями, который использовал кошелек Джона, когда впервые отправлял транзакцию Тому (рис. 8.14). В результате Лиза, Ци и Рашид получат сообщение от Тома.

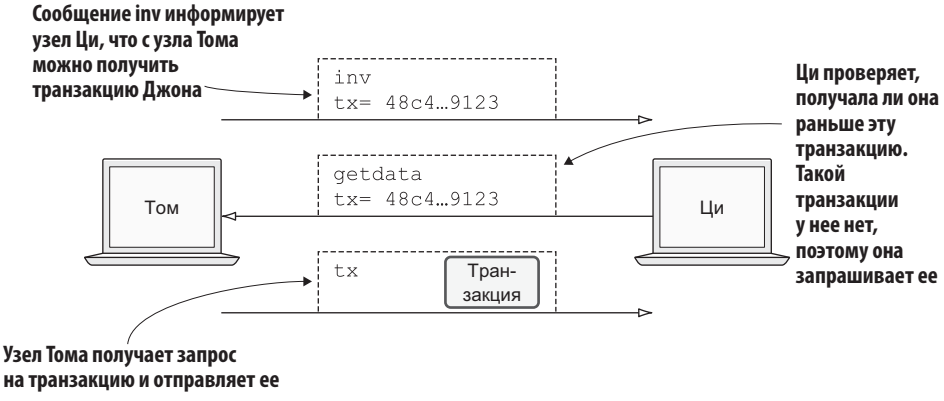
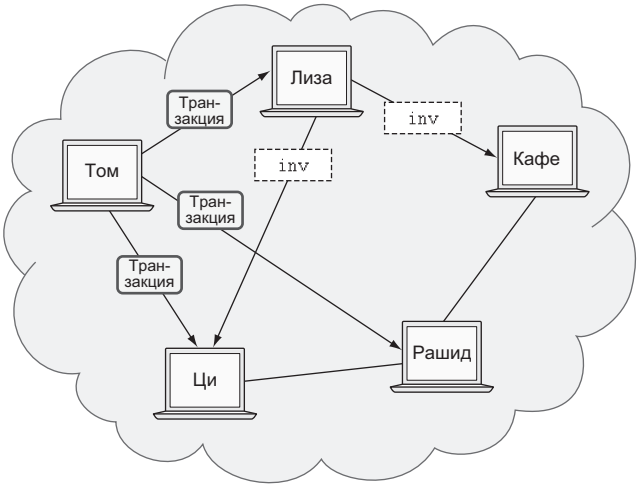


Рис. 8.14. Узел Тома посылает транзакцию узлу Ци, используя уже знакомую процедуру, состоящую из трех этапов

Когда Лиза, Ци и Рашид получат транзакцию, они также проверят ее и сообщат о ней своим соседям. Узлы Ци и Рашида немного медленнее, поэтому им требуется чуть больше времени для проверки транзакции; мы вернемся к ним позже.



Лиза быстрее всех проверит транзакцию и окажется первой из этой тройки, для кого настанет момент передать транзакцию дальше. Она уже знает, что получила транзакцию от Тома, поэтому не будет посылать ему сообщение `inv`. Но Лиза не знает, что Ци уже получила транзакцию, а также не знает, получил ли эту транзакцию узел в кафе. Она пошлет `inv` этим двум узлам.

Узел в кафе ответит запросом `getdata`, потому что еще не получал эту транзакцию. Узел Ци уже имеет эту транзакцию и поэтому ничего не ответит (рис. 8.15), но запомнит, что транзакция уже есть у Лизы.

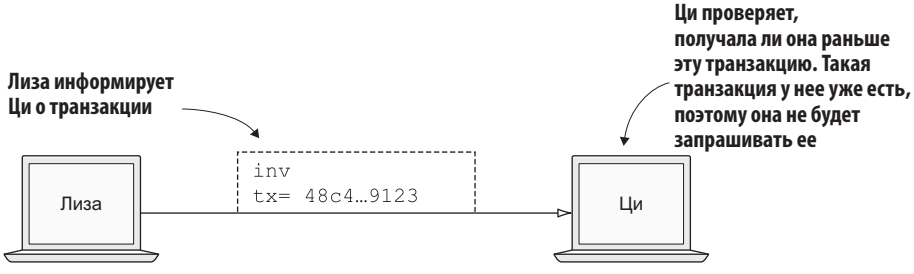
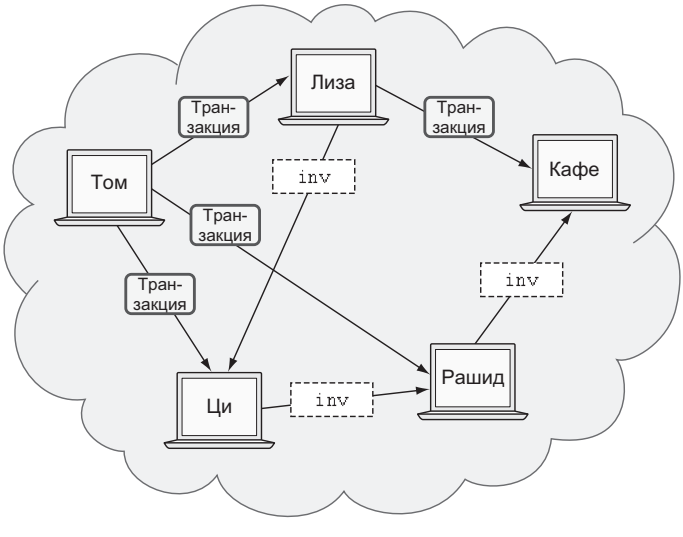


Рис. 8.15. Узел Лизы пошлет сообщение inv узлу Ци, но у него уже есть эта транзакция

Ци только что завершила проверку транзакции и знает, что она есть у узла Лизы, поэтому ей не нужно отправлять ему сообщение `inv`. Но она не знает, получал ли эту транзакцию Рашид, поэтому отправляет `inv` узлу Рашида.



Узел Рашида — самый медленный и позже всех завершил проверку транзакции Джона, поэтому, когда для него наступает момент отправить `inv` соседу, он уже получил `inv` от узла Ци. Еще раньше он узнал, что у Тома уже есть эта транзакция. Поэтому он отправит `inv` только узлу в кафе, который проигнорирует сообщение, потому что уже получил транзакцию.

Легкий кошелек кафе уведомляется о транзакции

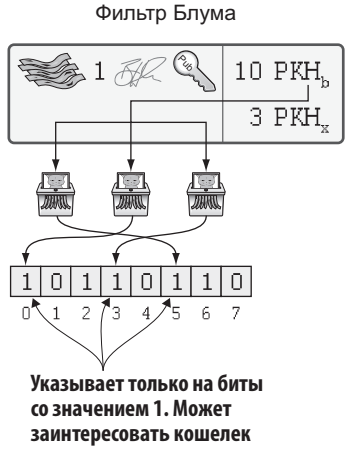
Как я уже говорил, передача транзакций через одноранговую сеть имеет одно замечательное свойство: кошелек получателя может быстро получить уведомление об ожидающей транзакции. Давайте посмотрим, как это происходит.

Полный узел в кафе получил транзакцию и проверил ее. В кафе также есть легкий кошелек на мобильном телефоне, который используется для отправки и получения денег. Владельца кафе волнует вопрос безопасности, поэтому он настроил легкий кошелек так, чтобы он подключался только к собственному полному узлу в кафе, который считается его *доверенным узлом* (рис. 8.16).

Рис. 8.16. Легкий кошелек в кафе подключается только к собственному полному узлу

Благодаря этой типичной конфигурации кафе получает безопасность полного узла в сочетании с гибкостью и мобильностью легкого кошелька. Я описал эту конфигурацию в разделе «Безопасность легких кошельков» в главе 6.

Полный узел в кафе только что проверил транзакцию Джона и теперь хочет проинформировать соседей о ней. Он имеет соединение с узлом Лизы, Рашида и легким кошельком кафе. Полный узел уже знает, что узлы Лизы и Рашида имеют эту транзакцию, поэтому он не отправляет `inv` этим двум узла. Он не



знает, есть ли транзакция в кошельке, но не будет сразу отправлять ему сообщение `inv`.

Кошелек в кафе — это легкий кошелек, использующий фильтры Блума, описанные в разделе «Фильтры Блума маскируют адреса» в главе 6. Полный узел проверит транзакцию на соответствие фильтру Блума и, если они совпадут, отправит в кошелек сообщение `inv`. Если транзакция не соответствует фильтру, сообщение не будет отправлено.

Транзакция Джона предназначена для кафе, поэтому она будет соответствовать фильтру Блума, и полный узел отправит `inv`. Кошелек запросит фактическую транзакцию с помощью `getdata`, как показано на рис. 8.17.

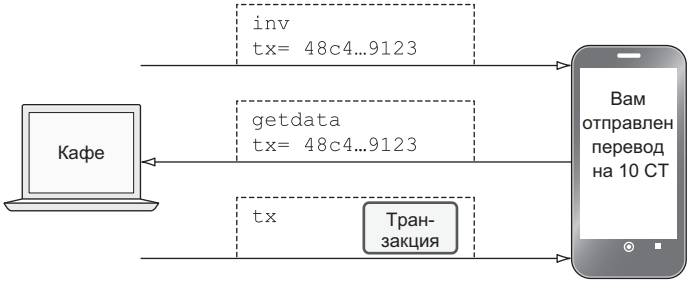


Рис. 8.17. Кошелек кафе получает транзакцию Джона от доверенного узла в кафе после ее сопоставления с фильтром Блума

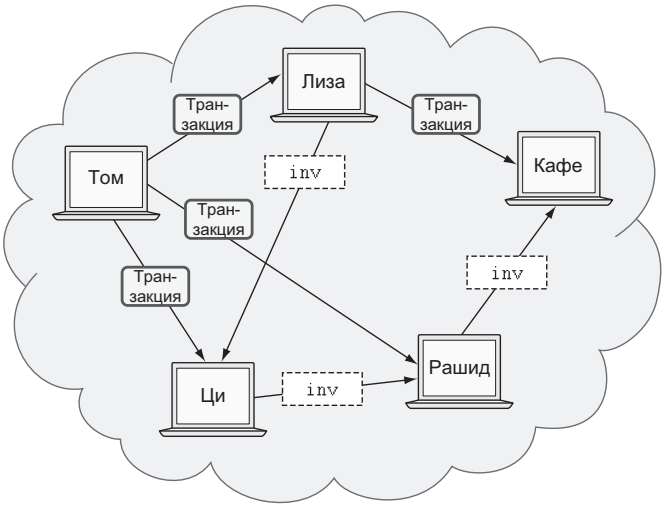
Получив транзакцию, кошелек может показать сообщение владельцу кафе, что транзакция ожидает подтверждения. У владельца кафе есть выбор: понадеяться, что транзакция — так называемая *транзакция с нулевым числом подтверждений* — будет в конечном счете подтверждена, или подождать ее фактического подтверждения. Если кафе примет транзакцию с нулевым числом подтверждений, значит, оно верит, что Джон заплатил достаточно высокую комиссию за транзакцию и средства не будут потрачены дважды.

Но давайте представим, что на этот раз в кафе решили подождать включения транзакции в действительный блок. Это подводит нас к следующему этапу: включению транзакции в блок в блокчейне.

Включение транзакции в блок

Давайте вспомним некоторых майнеров, участвовавших в работе этой системы. В конце раздела «Ограничение централизации майнинга» в главе 7 у нас

было 10 разных майнеров; но давайте вернемся назад и представим, что Ци, Том, Лиза и Рашид — единственные майнеры в этой системе.



Транзакция достигла всех этих майнеров на этапе ее распространения. Раньше кошелек Джона посылал транзакции по электронной почте. Теперь он отправляет их любому полному узлу, после чего они автоматически распространяются по всей одноранговой сети. Майнеры могут включить транзакцию Джона в свои блоки. Предположим, что транзакция содержит комиссию за транзакцию, поэтому некоторые или все майнеры готовы включить ее, и что Рашид первым находит действительное доказательство работы для своего блока, содержащего транзакцию Джона (рис. 8.18).

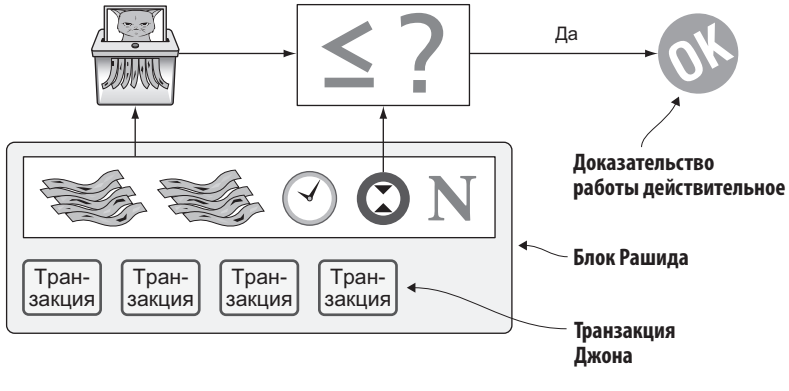


Рис. 8.18. Блок Рашида с транзакцией Джона

Рашид хочет как можно быстрее передать свой блок другим майнерам, чтобы минимизировать риск, что какой-то другой майнер создаст свой блок раньше.

Он создает сообщение `headers` и посылает его всем своим соседям: Тому, в кафе и Ци. Соседи Рашида вернут обратно сообщение `getdata`, и Рашид отправит им фактический блок. Обмен сообщениями между Рашидом и Ци будет выглядеть, как показано на рис. 8.19.

BIP 130

Эта процедура определена в стандарте BIP 130, заменившем старый механизм распространения блоков, использовавший сообщения `inv`.

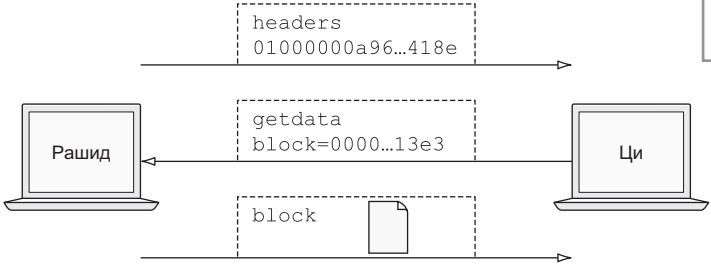
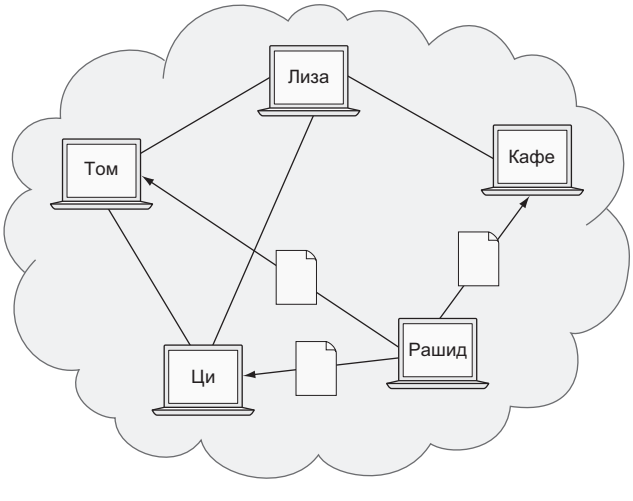


Рис. 8.19. Узел Рашида посылает блок узлу Ци

Фактический блок посылается в сообщении `block`.

Теперь посмотрим, как происходит распространение блоков в одноранговой сети. Рашид отправляет свой блок Тому, в кафе и Ци. Эти три узла проверят блок и, если он действительный, отправят сообщения `headers` всем своим соседям, которые могли еще не получить его (рис. 8.20).



Ци и Том одновременно отправят друг другу свои сообщения `headers`. Это не вызовет никаких проблем — поскольку оба узла уже получили блок, они просто проигнорируют эти сообщения. Лиза запросит блок у одного из своих соседей, точно так же как Ци запросила блок у Рашида.

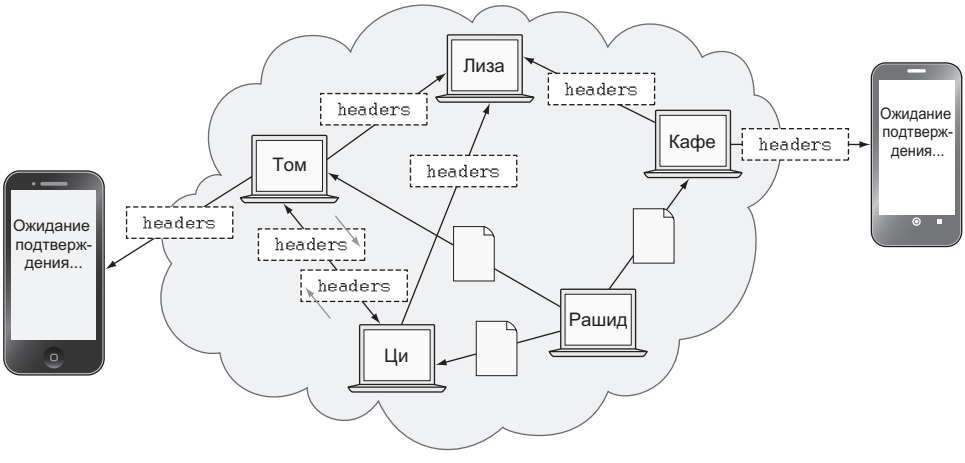


Рис. 8.20. Все, кроме Лизы, получили блок. Том, кафе и Ци посылают сообщение headers

На этом распространение блока завершается... почти. Осталось лишь проинформировать легкие кошельки о новом блоке.

Вывод уведомлений в кошельках

Узел Тома подключен к кошельку Джона, поэтому Том отправляет Джону сообщение `headers`. Аналогично полный узел кафе отправляет сообщение `headers` легкому кошельку кафе. Полные узлы Тома и кафе никак не проверяют блок на соответствие фильтру Блума. Сообщение `headers` отправляется безоговорочно, но легкие кошельки не будут запрашивать полные блоки.

Как рассказывалось в главе 6, легкие кошельки не загружают полные блоки. Чтобы проверить доказательство работы в блокчейне, кошельку Джона достаточно заголовка блока. Но иногда, когда транзакции, имеющие отношение к кошельку Джона, включаются в блоки, он должен получить доказательство такого включения. Чтобы выяснить наличие релевантных транзакций, кошелек посылает Тому сообщение `getdata`, запрашивая сообщение `merkleblock` для блока.

Джон получает сообщение `merkleblock` с заголовком блока и частичное дерево Меркла, связывающее идентификатор транзакции (txid) с корнем дерева Меркла в заголовке блока (рис. 8.21).

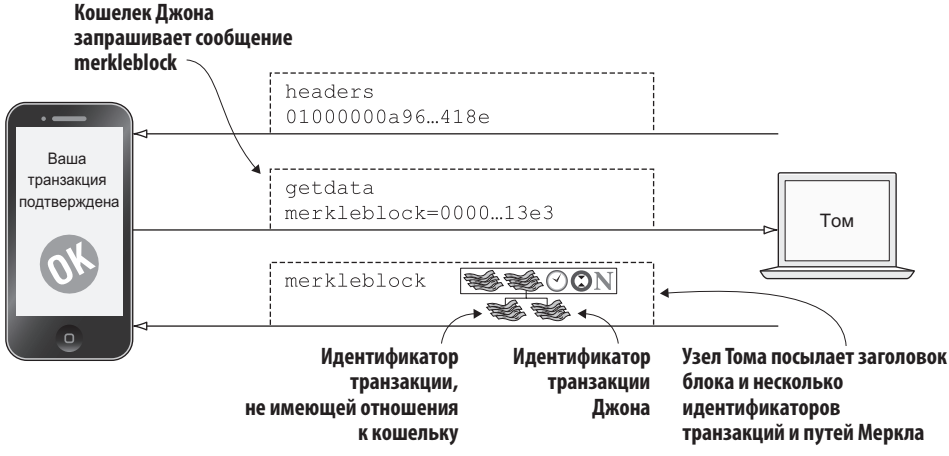


Рис. 8.21. Том посылает сообщение merkleblock с доказательством Меркла о включении транзакции Джона в блок

Рисунок 8.22 напоминает, о чем рассказывалось в главе 6.

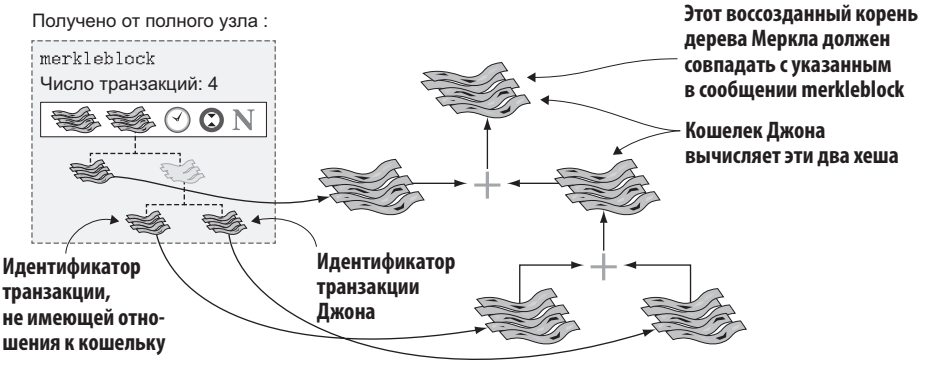
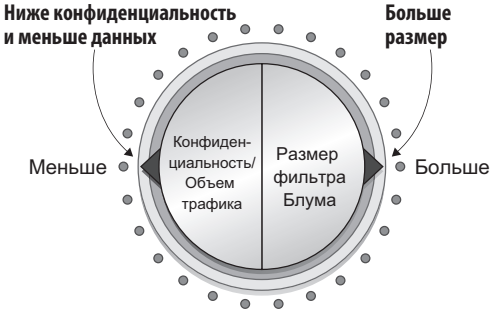


Рис. 8.22. Сообщение merkleblock содержит заголовок блока и частичное дерево Меркла

Кошелек Джона убедится, что:

- заголовок блока правильный и имеет действительное доказательство работы;

- корень дерева Меркла в заголовке можно восстановить, используя частичное дерево Меркла;
- идентификатор транзакции Джона включен в частичное дерево Меркла; его не интересуют транзакции, не имеющие отношения к кошельку и присланные для маскировки транзакции, принадлежащей Джону.



Убедившись, что транзакция включена в новый блок, кошелек Джона сможет отобразить сообщение Джону: «Ваша транзакция получила 1 подтверждение».

Легкий кошелек кафе выведет похожее уведомление.

Кошелек в кафе использует доверенный узел, поэтому сохранение конфиденциальности для него не является большой проблемой (рис. 8.23). Он может использовать большой фильтр для уменьшения мобильного трафика. Чем точнее фильтр Блума, тем меньше маскирующего трафика будет отправляться в кошелек.

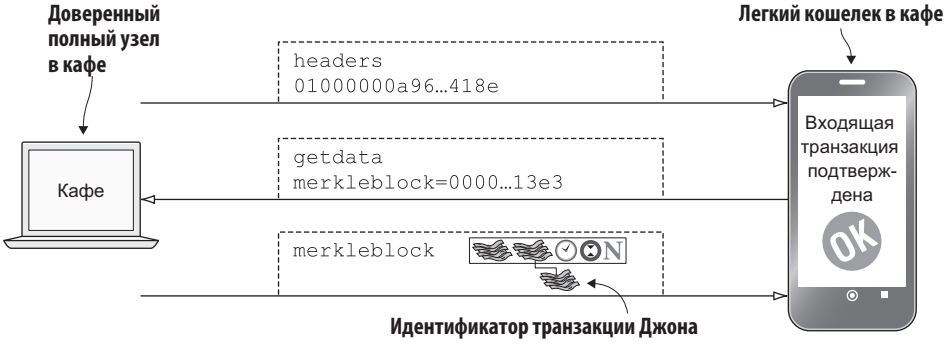


Рис. 8.23. Кошелек посылает запрос merkleblock доверенному полному узлу

Владелец кафе получает чувство уверенности и с удовольствием вручает булочку Джону. Джон кушает булочку. Сделка завершена.

Большее число подтверждений

С течением времени майнеры будут создавать новые блоки, которые распространятся по сети и, в конце концов, попадут на все полные узлы. Лег-

кие кошельки будут запрашивать сообщения `merkleblock` для экономии трафика.

С каждым новым блоком транзакция Джона будет скрываться под все большей толщей доказательств работы (рис. 8.24) и все труднее будет повторно потратить средства, которые она расходует. С каждым новым блоком транзакция будет получать еще одно подтверждение.

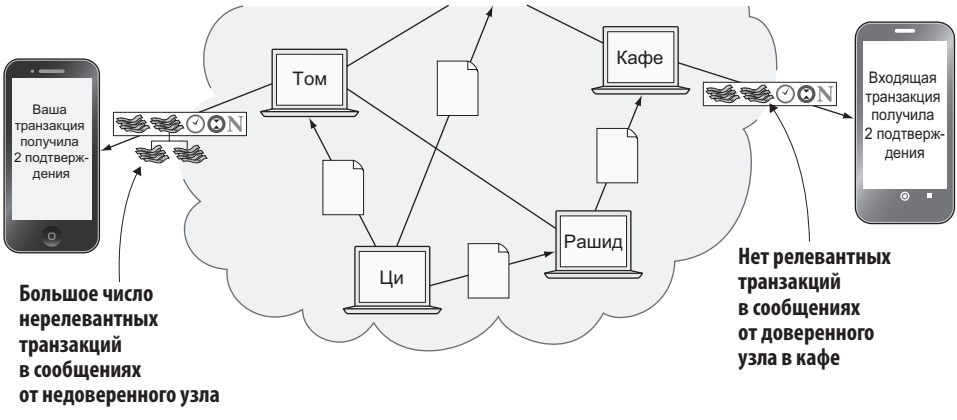


Рис. 8.24. С каждым новым блоком транзакция Джона становится все надежнее и надежнее

Оставляем в прошлом систему жетонов на булочки

Не думаю, что дальнейшее рассмотрение системы жетонов на булочки поможет нам лучше понять систему Биткоин. Пришло время оставить жетоны и булочки в прошлом и начать говорить исключительно о Биткоин. Мы развили систему жетонов на булочки до состояния, когда она перестала отличаться от Биткоин. В табл. 8.1 показано соответствие понятий в двух системах.

Таблица 8.1. На смену общей папке пришла одноранговая сеть

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2
~~Общая папка~~	~~Сеть Биткоин~~	~~Глава 8~~

Последним понятием, отличавшим системы жетонов на булочки и Биткоин, было понятие общей папки. Теперь мы избавились от него. Теперь взгляните на рис. 8.25, иллюстрирующий, как это получилось.

Задержим наших друзей в офисе еще ненадолго. Джон, вероятно, захочет купить еще несколько булочек, но теперь он будет использовать для этого систему Биткоин.

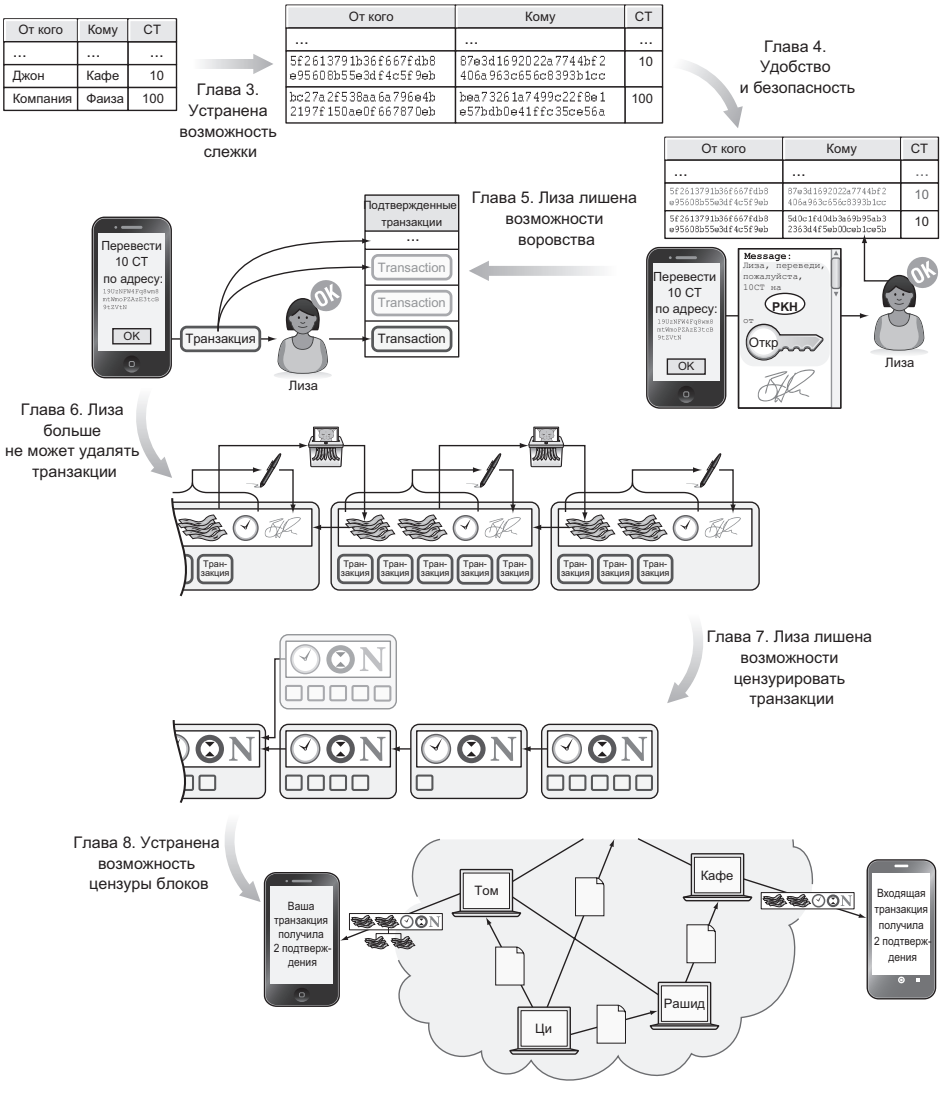


Рис. 8.25. Эволюция системы жетонов на булочки

Кратко о Биткоин

Сеть Биткоин огромна. Когда писалась эта книга:

- существовало примерно 10 000 общедоступных полных узлов;
- денежная масса в Биткоин составляла примерно 17 400 000 BTC;
- один биткоин стоил примерно $6500;
- ежедневно в сети Биткоин обрабатывалось около 250 000 транзакций;
- сумма ежедневных платежей оценивалась величиной в 100 000 BTC, или около 630 миллионов долларов;
- суммарный хешрейт майнеров составлял около 50 Эхеш/с, или 50×10^{18} хеш/с; типичный настольный компьютер имеет хешрейт около 25 Мхеш/с;
- ежедневные комиссионные отчисления за транзакции оценивались суммой в 17 BTC, то есть в среднем 6800 сатоши или около $0,40 за транзакцию;
- люди по всему миру пользовались системой Биткоин для решения своих повседневных проблем.

На чем мы остановились?

Эта глава посвящена одноранговой сети Биткоин. Первая половина главы описывала работу этой сети после настройки, как показано на рис. 8.26, повторяющем рисунок из главы 1.

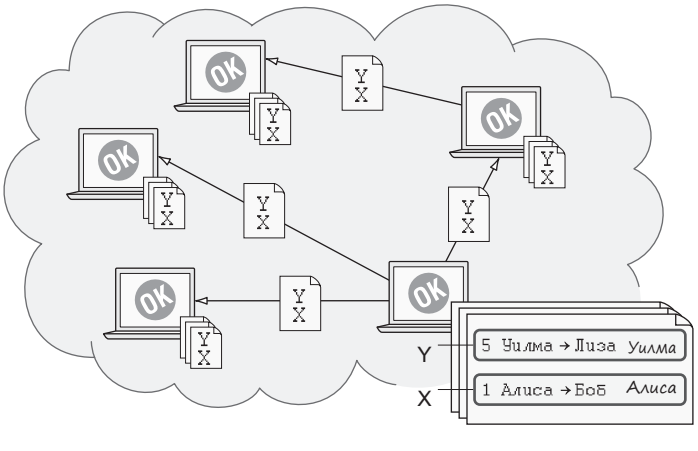


Рис. 8.26. Сеть Биткоин распространяет блоки (и транзакции) между всеми участниками

Во второй половине этой главы мы посмотрим, как новый узел присоединяется к сети.

Инициализация сети

Сценарий, описанный в разделе «Сетевой протокол», предполагал, что все действующие узлы уже подключены друг к другу. Но как запустить новый узел? Как он отыщет другие узлы для подключения? Как загрузит полный блокчейн, начиная с первичного блока, с порядковым номером 0, до самого последнего? Как он узнает, какой блок последний?

Давайте разберемся.

Предположим, что Селма решила запустить свой полный узел. Вот как это обычно происходит (рис. 8.27):

❶ Селма загружает, проверяет и запускает программное обеспечение полного узла.

❷ Это программное обеспечение соединяется с некоторыми другими узлами.

❸ Узел Селмы загружает блоки от своих соседей.

❹ Узел Селмы переходит в режим нормальной работы.

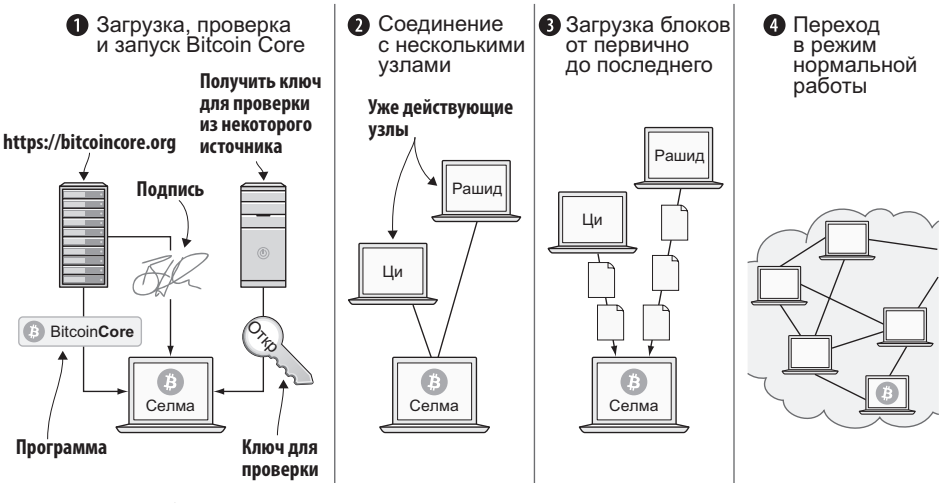


Рис. 8.27. Чтобы запустить полный узел, нужно загрузить и запустить программное обеспечение, соединиться с другими узлами, загрузить старые блоки и перейти в режим нормальной работы

Шаг ❶ — запуск программного обеспечения

Чтобы запустить полный узел, Селме нужна компьютерная программа. Чаще всего для этих целей используется программа Bitcoin Core. Также есть несколько других программ, таких как libbitcoin, bcoin, bitcoinj и btcd, но мы сосредоточимся только на Bitcoin Core, а с остальными вы можете познакомиться самостоятельно.

Чтобы загрузить Bitcoin Core, Селма открыла веб-страницу https://bitcoincore.org и нашла там ссылку для скачивания. Но столкнулась с потенциальной проблемой: Селма не уверена, что программа, которую она загружает, действительно создана разработчиками Bitcoin Core. Кто-то мог обмануть Селму, подсунув ей программу с сайта bitconcore.org вместо bitcoincore.org, или взломать bitcoincore.org и заменить загружаемые файлы фейковыми программами.

Чтобы исключить подобные проблемы, команда Bitcoin Core подписывает все выпущенные ими версии программы закрытым ключом — назовем его *ключом Bitcoin Core*. Они предоставляют подпись в загружаемом файле с именем SHA256SUMS.asc. Этот файл содержит хеш программы Bitcoin Core и подпись содержимого файла SHA256SUMS.asc (рис. 8.28).

Рис. 8.28. Команда Bitcoin Core подписывает выпущенные ими версии программы своим закрытым ключом

Селма загружает программу, файл с именем bitcoin-0.17.0-x86_64-linux-gnu.tar.gz, и файл подписи SHA256SUMS.asc. Чтобы убедиться, что программа действительно подписана закрытым ключом Bitcoin Core, ей необходимо получить соответствующий открытый ключ. Но где его взять?

Это сложная проблема. Помните, как раньше Лиза подписывала блоки своим закрытым ключом? Как в ту пору полные узлы могли проверить, что блоки действительно были подписаны Лизой? Они могли получить открытый ключ Лизы из нескольких источников — например, с доски объявлений у входа в офис, из внутренней сети компании и у коллег. То же самое и здесь; не доверяйте одному источнику и постарайтесь получить ключ как минимум из двух разных источников. Ключ, который в настоящее время используется для подписания версий Bitcoin Core, называется

```
Wladimir J. van der Laan (Bitcoin Core binary release signing key)
↳<laanwj@gmail.com>
```

и имеет следующий 160-битный хеш SHA1, который называется *отпечатком*:

```
01EA 5486 DE18 A882 D4C2 6845 90C8 019E 36C2 E964
```

Этот блок может служить *одним из* источников для Селмы. Она решает:

1. Взять отпечаток ключа на сайте https://bitcoincore.org.
2. Взять отпечаток из книги «Грокаем технологию Биткоин».
3. Взять отпечаток у друзей.

Отпечатки из всех трех источников совпадают, поэтому Селма загружает открытый ключ с *сервера ключей*. Сервер ключей — это компьютер в Интернете, который выполняет функцию хранилища ключей. Серверы ключей обычно используются для загрузки ключей, идентифицируемых по отпечаткам. Селма не доверяет серверу ключей, поэтому она проверяет соответствие отпечатка загруженного ключа отпечатку, полученному из нескольких источников, и убеждается в его действительности.

ГДЕ ВЗЯТЬ КЛЮЧ

Неважно, откуда вы получите настоящий открытый ключ, важно, чтобы его отпечаток соответствовал ожидаемому.

Теперь, имея открытый ключ Bitcoin Core, она может проверить подпись в файле SHA256SUMS.asc (рис. 8.29).

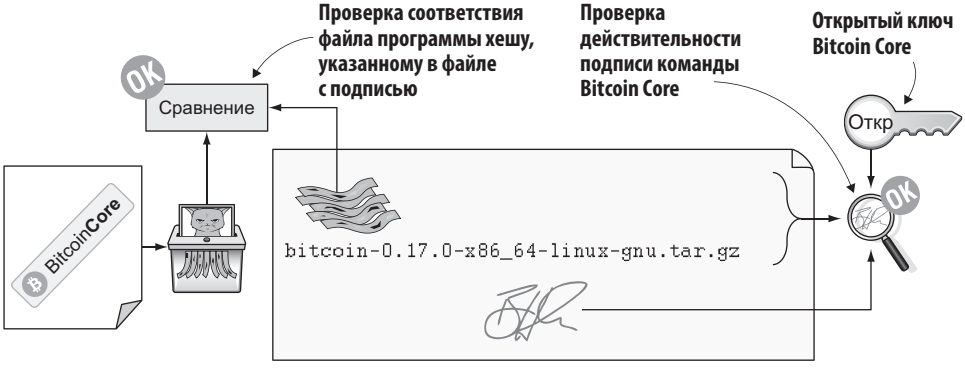
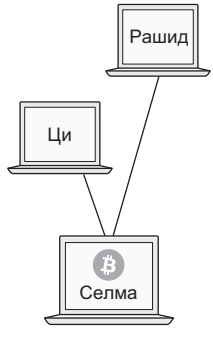


Рис. 8.29. Селма проверяет подпись Bitcoin Core и совпадение хеша в файле с хешем фактической программы

Она должна использовать открытый ключ Bitcoin Core для проверки подписи в файле SHA256SUMS.asc и убедиться, что программа имеет такое же значение хеша, как указано в SHA256SUMS.asc. Подпись действительна, хеши совпадают, и теперь Селма может быть уверена в подлинности программного обеспечения, которое собирается запустить.

Селма запускает программу на своем компьютере.

Шаг ❷ — соединение с узлами



После запуска программа полного узла на компьютере Селмы не подключается ни к каким другим узлам. Она пока не является частью сети Биткоин. На этом этапе узел попытается найти соседние узлы для подключения.

Для подключения к другому полному узлу необходимо знать его IP-адрес и номер порта TCP. Например:

```
IP: 142.12.233.96 port: 8333
```

IP-адрес и номер порта обычно записываются вместе:

```
142.12.233.96:8333
```

Поиск соседних узлов

Где узел Селмы сможет найти адреса других узлов? Есть несколько источников этой информации (рис. 8.30):

- Адреса соседей могут быть перечислены в настройках полного узла. Эти адреса Селма может узнать у своего друга, уже имеющего полный узел.
- Адреса можно получить с помощью системы доменных имен (Domain Name System, DNS).
- Адреса могут быть «жестко зашиты» в код программы полного узла.

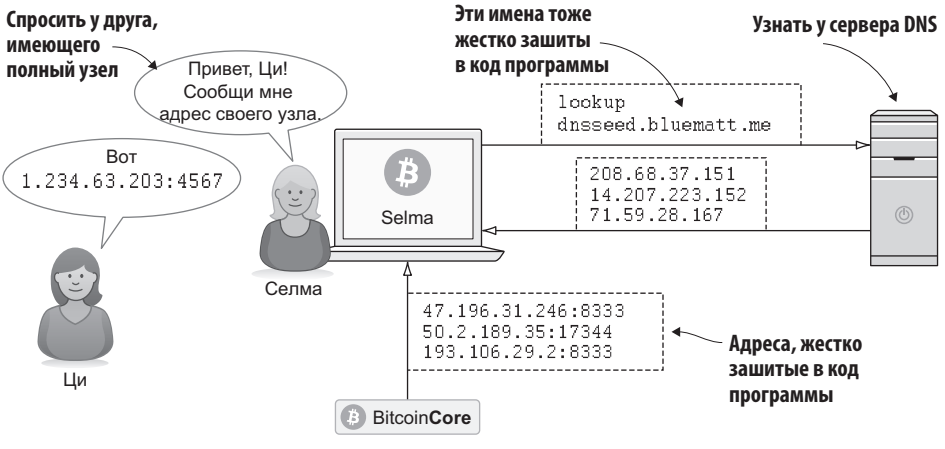
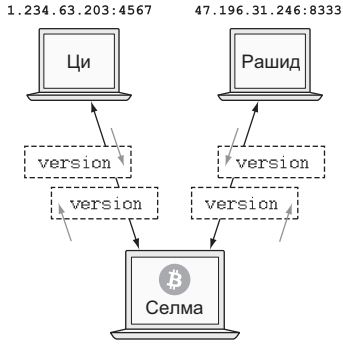


Рис. 8.30. Полный узел Селмы имеет три разных источника для получения адресов начальных соседей

Узел Селмы должен соединиться не с одним, а сразу с несколькими узлами. Если соединиться только с одним узлом, он может оказаться мошенническим, и она не узнает об этом. Если сразу соединиться с несколькими узлами, можно убедиться, что все они отправляют друг другу непротиворечивые данные. Если это не так, значит, один или несколько узлов намеренно искажают информацию или сами подверглись обману.

По умолчанию для поиска адресов начальных узлов используется система DNS. DNS — это глобальная система имен, используемая для поиска IP-адресов по именам компьютеров. Например, когда вы открываете страницу *https://bitcoin.org* в веб-браузере, он обращается к системе DNS, чтобы определить IP-адрес по имени *bitcoin.org*. Программное обеспечение Bitcoin Core делает то же самое. Имена для поиска жестко зашиты в Bitcoin Core, так же как некоторые IP-адреса и порты. Поиск в DNS может вернуть несколько IP-адресов, и каждая новая попытка поиска может возвращать другой набор IP-адресов. Этот, третий, вариант используется как последнее средство.

Обратите внимание, что запросы к DNS, изображенные на рис. 8.30, не возвращают номера портов. Два других метода поиска начальных соседей обычно включают их, но сервер DNS может возвращать только IP-адреса. Предполагается, что узлы с этими IP-адресами прослушивают порт, используемый в Bitcoin Core по умолчанию, а именно 8333.



Рукопожатие

Допустим, что Селма решила подключиться к узлу Ци с адресом 1.234.63.203:4567 и к узлу Рашида с адресом 47.196.31.246:8333. Селма настраивает TCP-соединение с каждым из этих двух узлов и посылает им начальное сообщение. Давайте посмотрим, как протекает общение с узлом Ци (рис. 8.31).

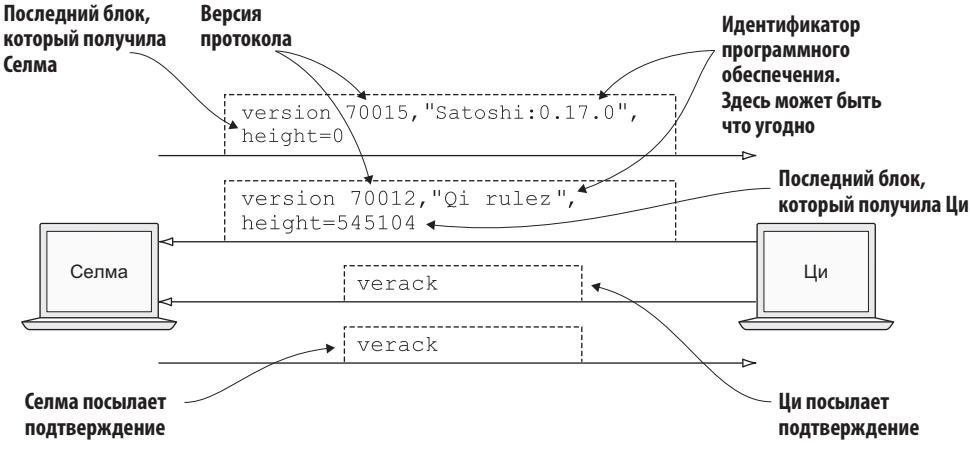


Рис. 8.31. Селма обменивается с Ци сообщениями version

Этот обмен, называемый *рукопожатием*, начинает Селма. Она отправляет сообщение `version` узлу Ци. Рукопожатие используется, чтобы согласовать версию протокола и сообщить друг другу, какую высоту блоков они имеют. Сообщение `version` содержит много сведений, не показанных на рис. 8.31, из которых наиболее важными являются:

- *Версия протокола* — версия сетевого протокола, или «языка», который используют узлы для общения друг с другом. Селма и Ци будут использовать

версию 70012, потому что это самая высокая версия, которую понимает узел Qi. Селма знает все версии протокола вплоть до собственной (70015).

- *Пользовательский агент* — на рис. 8.31 эта информация обозначена как «идентификатор программного обеспечения», потому что название «пользовательский агент» выглядит малопонятным. Эта информация используется, чтобы подсказать другому узлу, какое программное обеспечение использует данный узел, но вообще это может быть что угодно.
- *Высота* — высота последнего блока в сильнейшей цепочке, полученного узлом.

Также сообщение `version` включает другую полезную информацию:

- *Службы* — список возможностей, поддерживаемых узлом, таких как фильтрация Блума для легких кошельков.
- *Мой адрес* — IP-адрес и номер порта узла, отправившего сообщение `version`. Без этого Ци не узнала бы адрес узла для повторного подключения, если бы решила перезагрузить компьютер и вновь соединиться с узлом Селмы.

Получив сообщение `version` от узла Селмы, узел Ци ответит своим сообщением `version`. Он также отправит сообщение `verack` сразу после сообщения `version`. Сообщение `verack` не содержит никакой информации — оно используется для подтверждения, что Ци получила сообщение `version`.

Получив сообщение `version` от узла Ци, узел Селмы ответит своим сообщением `verack`. На этом этап рукопожатия завершается. Далее узел Селмы повторяет процедуру рукопожатия с узлом Рашида.

Поиск соседей соседей

Соединившись с узлом Рашида, узел Селмы запросит у него адреса других узлов для подключения. Так Селма может расширить свой круг соседей (рис. 8.32).

Селма установила связь только с двумя соседями: узлами Ци и Рашида. Но ей хотелось бы связаться с большим числом узлов. Связь лишь с двумя узлами может иметь некоторые неприятные последствия:

- Ци и Рашид могут договориться скрывать транзакции и блоки от Селмы.
- Узел Ци может выключиться, и тогда у Селмы останется связь только с узлом Рашида. В этом случае Рашид сможет самостоятельно скрывать информацию от Селмы.

✶ Узлы Ци и Рашида оба могут выключиться, и тогда Селма окажется полностью отключенной от сети, пока не подключится к другим узлам с помощью механизма начального поиска соседей.

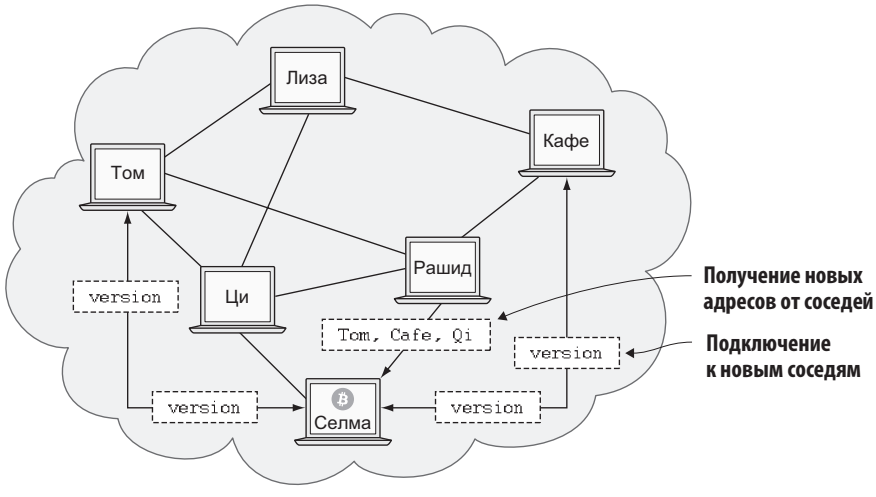


Рис. 8.32. Селма запрашивает у своих соседей адреса других соседей, чтобы связаться с ними

На рис. 8.33 показано, как Селма запрашивает у Рашида адреса других соседей.

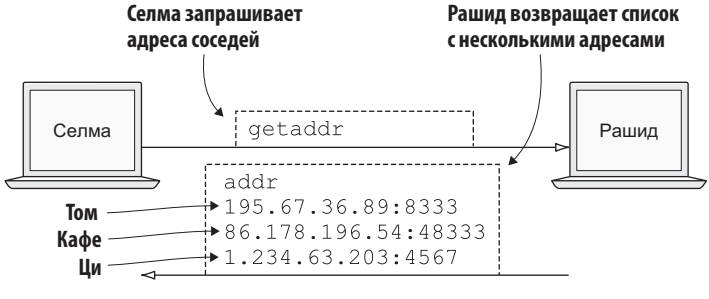


Рис. 8.33. Селма запрашивает у Рашида адреса других узлов. Он возвращает ей целый список

Селма посылает узлу Рашида сообщение `getaddr`. В ответ Рашид возвращает список IP-адресов и портов TCP, которые Селма может использовать для соединения с большим числом соседей. Какие адреса послать Селме, решает Рашид, но обычно в список включаются адреса узлов, с которыми Рашид

уже соединился и, возможно, которые получил от своих соседей, но не использовал сам.

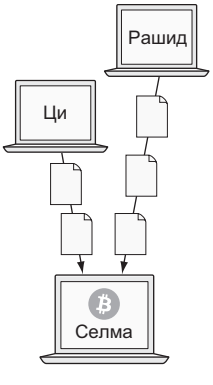
НАЧАЛЬНЫЕ УЗЛЫ

После получения сообщения `addr` узлы отключаются от начальных узлов (кроме настроенных вручную), чтобы не перегружать их. Они играют роль начальных узлов для многих других.

Селма соединяется со всеми или некоторыми адресами из списка, чтобы расширить свой *круг общения*. Чем к большему числу соседей вы подключены, тем шире круг. Широкий круг общения снижает риск пропустить информацию из-за ненадлежащего поведения соседей. Кроме того, чем шире круг общения, тем быстрее распространяется информация. Типичный полный узел в Биткоин поддерживает около 100 активных соединений. Только восемь (по умолчанию) из них являются *исходящими соединениями*, то есть соединениями, инициированными данным узлом. Остальные являются *входящими соединениями*, инициированными другими узлами. Соответственно, полный узел, недоступный через порт 8333 из Интернета — например, из-за особенностей настроек брандмауэра, — в общем случае будет иметь не больше восьми соединений.

Шаг ❸ — синхронизация

Теперь, установив широкий круг общения в сети Биткоин и став ее частью, Селма приступает к загрузке и проверке всего блокчейна, до самого последнего доступного блока. Этот процесс называется *синхронизацией*, или *начальной загрузкой блокчейна*.



Изначально Селма имеет только один блок: первичный (генезис-блок). Первичный блок жестко зашит в программное обеспечение Bitcoin Core, поэтому все узлы имеют его сразу после запуска.

Она должна загрузить всю цепочку блоков от своих соседей и проверить их, прежде чем начать проверять вновь созданные блоки. Это необходимо, потому что она понятия не имеет, как выглядит текущий набор неизрасходованных выходов транзакций (Unspent Transaction Outputs, UTXO). Чтобы получить текущий набор UTXO, она должна начать с пустого набора UTXO, пройти по всем историческим блокам, начиная с блока 0, и на каждом шаге обновить набор UTXO, используя информацию из транзакций в блоках.

Вот как выглядит этот процесс:

1. С одного из узлов загружаются заголовки всех блоков и выполняется проверка доказательства работы.
2. Параллельно с нескольких соседних узлов загружаются все блоки из сильнейшей цепочки.

Селма выбирает одного из своих соседей, Тома, и загружает все заголовки блоков. На рис. 8.34 показано, как узел Селмы загружает заголовки блоков с узла Тома.

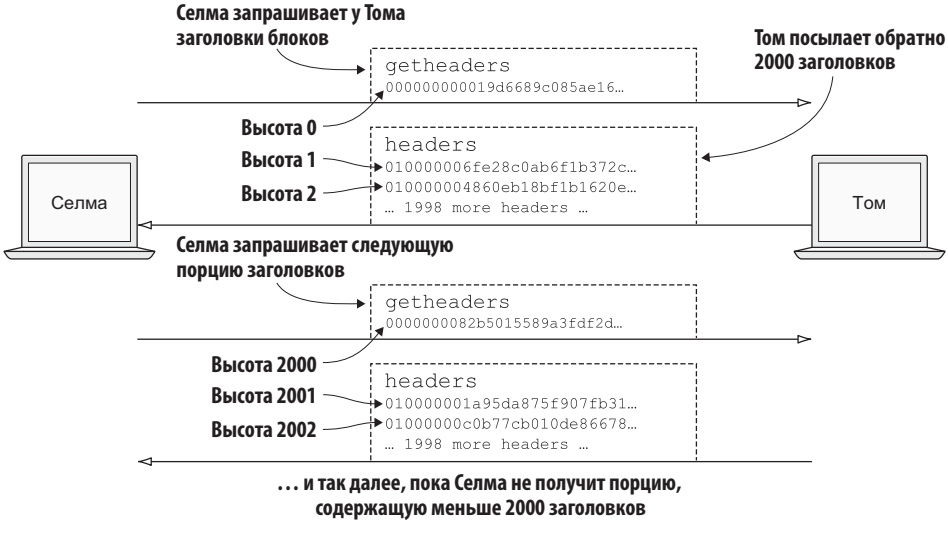


Рис. 8.34. Селма загружает заголовки блоков, продолжая снова и снова посылать Тому сообщение getheaders с идентификатором последнего блока, имеющегося у нее

УТОЧНЕНИЕ

Сообщение `getheaders` содержит список некоторых идентификаторов блоков из блокчейна Селмы, чтобы Том мог найти блок, имеющийся у них обоих. Это необходимо на тот случай, если у Тома не окажется блока, искомого Селмой. Но не будем пока беспокоиться об этом.

Она посылает сообщение `getheaders` с идентификатором последнего блока, имеющегося у Селмы, — первичного блока с порядковым номером 0. В от-

вет Том возвращает список, содержащий 2000 заголовков блоков, каждый из которых имеет размер 80 байт. Селма проверяет доказательство работы в каждом заголовке и запрашивает новую порцию заголовков у Тома. Этот процесс продолжается до тех пор, пока Селма не получит от Тома порцию, содержащую меньше 2000 заголовков, что свидетельствует о том, что у него больше нет заголовков для отправки ей.

Когда Селма получит все заголовки от Тома, она определит, какая ветвь является самой сильной, и начнет загружать фактические данные для блоков, принадлежащих этой ветви, с соседних узлов. Для ускорения процесса загрузка может выполняться из нескольких узлов одновременно. На рис. 8.35 показано, как происходит загрузка с узла Рашида.

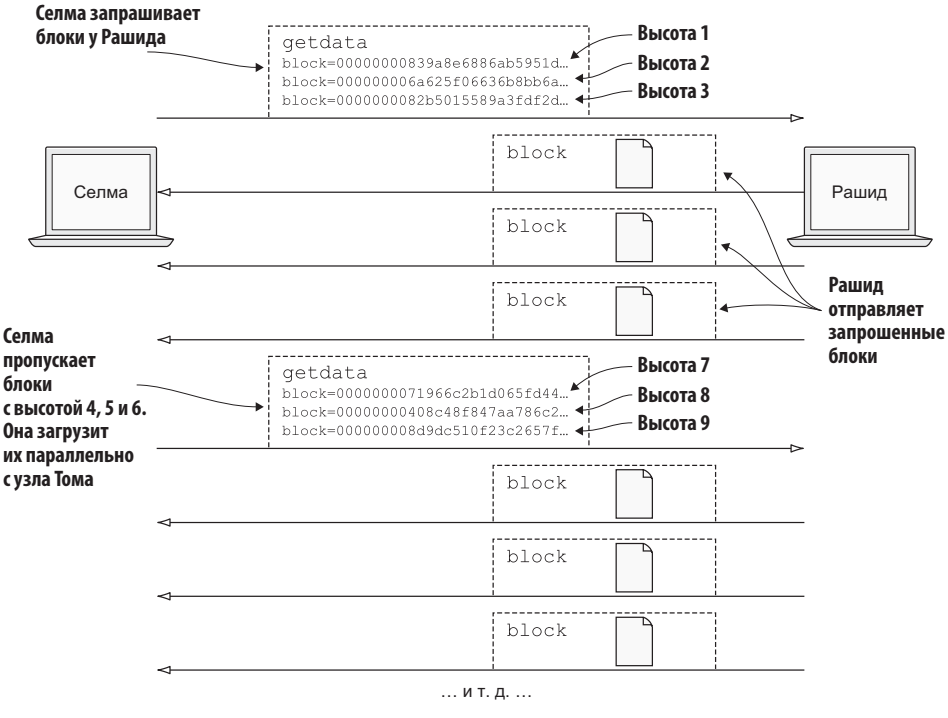


Рис. 8.35. Селма загружает блоки с узла Рашида, снова и снова посылая сообщение getdata со списком идентификаторов блоков

Загрузка начинается с того, что Селма отправляет Рашиду сообщение `getdata`. Это сообщение указывает, какие блоки она хочет загрузить. В ответ Рашид отправляет запрошенные блоки в сообщениях `block`, один за другим. Обратите внимание, что Селма загружает с узла Рашида не все

блоки. Часть блоков загружается параллельно с узла Тома, поэтому в последовательности запрашиваемых блоков есть пробелы. Процесс повторяется, пока Селма не решит прекратить загружать блоки с узла Рашида.

НАИБОЛЬШИЙ РАЗМЕР ПАКЕТА

В этом примере Селма запрашивает сразу 3 блока, но в действительности Bitcoin Core может запросить список, содержащий до 16 блоков.

Пока Селма загружает блоки, Рашид, вероятно, получит новые блоки от своих соседей. Предположим, он получил новый блок к тому времени, когда Селма загрузила первые 100 блоков. В этом случае Рашид отправит сообщение `headers` своим соседям, включая Селму, как описано в разделе «Включение транзакции в блок». Благодаря этому Селма узнает о всех новых блоках, появляющихся в течение времени, пока она выполняет начальную синхронизацию, и позже сможет запросить их у любого соседа.

НАЧАЛЬНАЯ ЗАГРУЗКА

На момент написания этой книги для начальной инициализации блокчейна требовалось загрузить около 210 Гбайт. Получение такого объема данных может занять несколько часов и даже дней, в зависимости от производительности оборудования и скорости подключения к Интернету.

Получая блоки, Селма проверяет их, обновляет свой набор UTXO и добавляет в свою цепочку блоков.

Проверка ранних блоков

Самая трудоемкая часть в проверке блока — проверка подписей в транзакциях. Зная, что идентификатор блока является частью действительной цепочки блоков, можно пропустить проверку подписей в этом и во всех предшествующих блоках (рис. 8.36). Это значительно ускорит начальную загрузку блокчейна до данного блока.

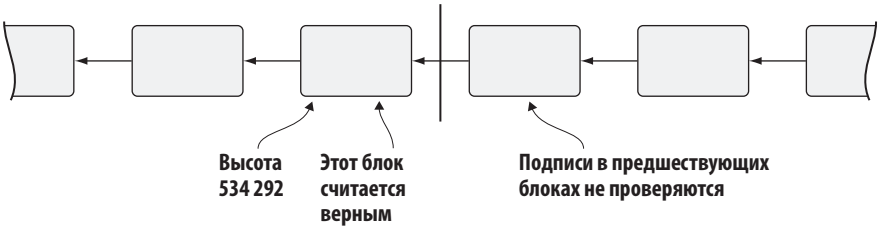


Рис. 8.36. Чтобы ускорить начальную загрузку блоков, подписи в достаточно старых блоках можно не проверять

Однако другие проверки, такие как отсутствие двойного расходования одних и тех же средств и правильная величина вознаграждения за блок, все еще выполняются. В ходе синхронизации узел должен создать свой набор UTXO, поэтому он в любом случае должен пройти через все транзакции и сконструировать набор UTXO.

В Bitcoin Core уже зашит идентификатор блока, включенный в блокчейн за несколько недель до даты выпуска программного обеспечения. Для Bitcoin Core 0.17.0 этот блок

```
height: 534292
hash: 00000000000000000002e63058c023a9a1de233554f28c7b21380b6c9003f36a8
```

На дату выпуска этой версии за данным блоком в блокчейн было включено примерно 10 000 блоков. Разумеется, это всего лишь параметр настройки, и упомянутый блок просто представляет разумное значение по умолчанию. Селма может изменить эту настройку перед запуском своего узла, узнав у друзей или из других источников, которым доверяет, какой блок можно с уверенностью считать представителем «цепочки всех действительных транзакций». Она также может отключить функцию проверки подписей всех транзакций, начиная с блока 0.

Через некоторое время Селма наконец оказывается на той же стадии, что и другие узлы, и готова войти в режим нормальной работы.

Шаг ❹ — режим нормальной работы

Этот шаг прост и уже обсуждался в разделе «Сетевой протокол». Узел Селмы переходит в режим нормальной работы. Отныне он будет участвовать в распространении блоков и транзакций и проверять каждую новую транзакцию и блок (рис. 8.37).

Теперь у Селмы есть полноценный полный узел.

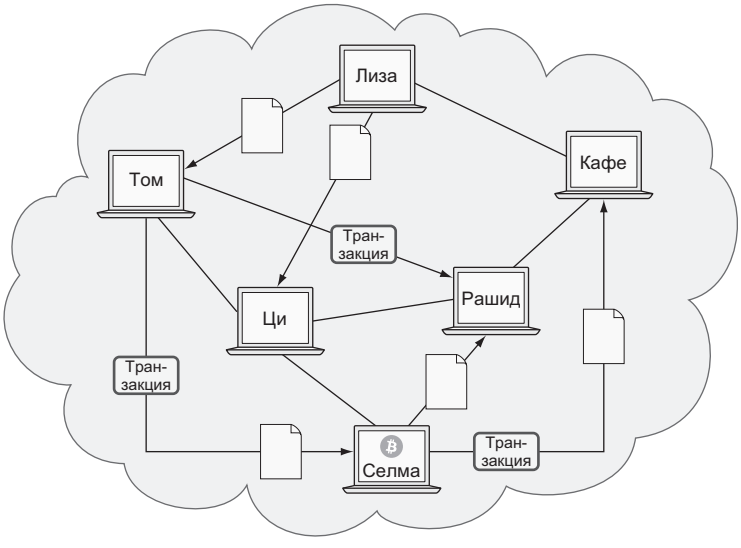


Рис. 8.37. Селма стала активным участником одноранговой сети Биткоин

Запуск собственного полного узла

ВНИМАНИЕ

В этом разделе описана настройка собственного полного узла Bitcoin Core в ОС Linux. Он предназначен для читателей, знакомых с ОС Linux и командной строкой.

Теперь вы имеете теоретическое представление, как загрузить, запустить и синхронизировать полный узел Биткоин. Этот раздел поможет вам установить собственный полный узел.

В этом разделе вам понадобятся:

- компьютер с 2 Гбайт ОЗУ или больше, действующий под управлением ОС Linux;
- большой объем свободного дискового пространства; на момент написания этих строк требовалось около 210 Гбайт;
- безлимитное подключение к Интернету;
- умение запускать и использовать терминал.

ДОПОЛНИТЕЛЬНЫЕ ИНСТРУКЦИИ

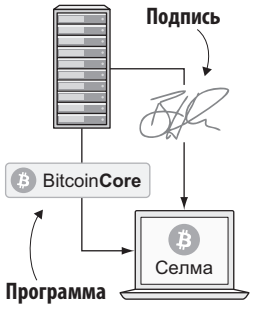
Инструкции для других основных операционных систем доступны на веб-ресурсе 18 (приложение B).

Если вы пользуетесь другой ОС, то все равно сможете использовать эти инструкции; для этого придется установить версию Bitcoin Core для вашей системы, и команды будут выглядеть по-другому. Изучите веб-ресурс 18 (приложение В) — там вы найдете самые свежие инструкции для своей ОС.

В общих чертах процесс запуска своего полного узла выглядит так:

1. Загрузить Bitcoin Core с сайта https://bitcoincore.org/en/download.
2. Проверить программное обеспечение.
3. Распаковать и запустить.
4. Дождаться окончания начальной загрузки блокчейна.

Загрузка Bitcoin Core



Чтобы запустить свой полный узел Биткоин, вам понадобится программа. В этом примере описывается загрузка Bitcoin Core с веб-ресурса 19 (Приложение В). На момент написания этой книги последней была версия Bitcoin Core 0.17.0. Загрузим ее:

```
$ wget https://bitcoincore.org/bin/bitcoin-core-0.17.0/\
    bitcoin-0.17.0-x86_64-linux-gnu.tar.gz
```

Как легко догадаться по имени файла bitcoin-0.17.0-x86_64-linux-gnu.tar.gz, команда загрузит версию 0.17.0 для 64-разрядной (x86_64) операционной системы Linux (linux-gnu). К тому моменту, когда вы будете читать эти строки, наверняка появится новая версия Bitcoin Core. Загляните на веб-ресурс 19, чтобы получить самую свежую версию Bitcoin Core. Кроме того, если вы используете другую операционную систему или компьютер с другой аппаратной архитектурой, выберите файл, подходящий для вас.

Проверка программного обеспечения

ВНИМАНИЕ

Это сложный раздел и требует выполнения большого числа операций в командной строке. Если вы просто хотите установить и запустить программное обеспечение Bitcoin Core для ознакомления и экспериментов, можете пропустить этот раздел и перейти к разделу «Распаковка и запуск». Если вы решили использовать его для майнинга, оцените риски, описанные выше, в разделе «Шаг ❶ — запуск программного обеспечения», прежде чем пропустить этот раздел.

В этом разделе рассказывается, как проверить достоверность загруженного файла .tar.gz. Этот файл подписан закрытым ключом команды Bitcoin Core. Процесс проверки включает следующие этапы:

1. Загрузка файла подписи.
2. Проверка совпадения хеша файла .tar.gz с хешем в файле подписи.
3. Загрузка открытого ключа команды Bitcoin Core.
4. Установка открытого ключа на свой компьютер.
5. Проверка подписи.

А теперь приступим.

Загрузка файла подписи

Чтобы убедиться, что загруженный пакет Bitcoin Core действительно принадлежит команде Bitcoin Core, необходимо загрузить файл подписи с именем SHA256SUMS.asc. На рис. 8.38, который уже приводился в разделе «Шаг ❶ — запуск программного обеспечения», показано, как устроен файл SHA256SUMS.asc.

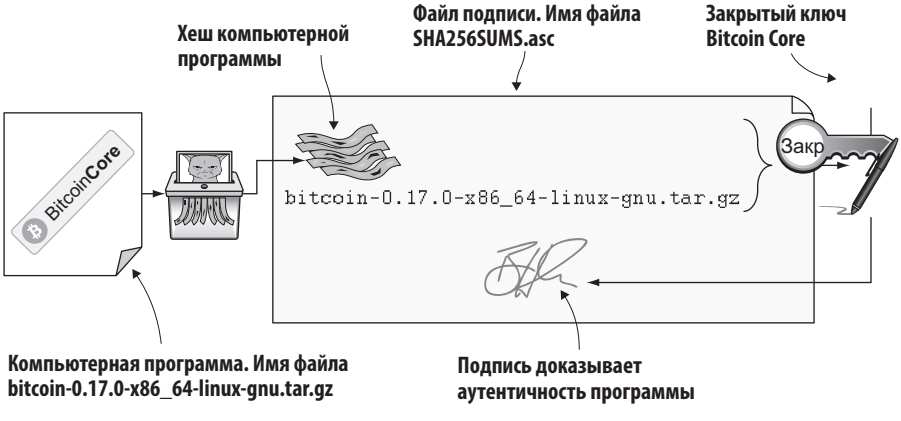


Рис. 8.38. Команда Bitcoin Core подписывает выпущенные версии программы своим закрытым ключом

Загрузите файл подписи SHA256SUMS.asc с того же сервера, с которого загрузили программу:

```
$ wget https://bitcoincore.org/bin/bitcoin-core-0.17.0/SHA256SUMS.asc
```

Этот файл мы используем, чтобы убедиться, что загруженный файл *.tar.gz* действительно подписан командой Bitcoin Core. Обратите внимание, что

этот файл предназначен только для версии 0.17.0. Для проверки другой версии Bitcoin Core следует выбрать соответствующий файл подписи на веб-ресурсе 19.

В следующем листинге приводится содержимое этого файла (хеши в этом листинге сокращены):

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

1e43...35ed bitcoin-0.17.0-aarch64-linux-gnu.tar.gz
a4ff...7585 bitcoin-0.17.0-arm-linux-gnueabihf.tar.gz
967a...f1b7 bitcoin-0.17.0-i686-pc-linux-gnu.tar.gz
e421...5d61 bitcoin-0.17.0-osx64.tar.gz
0aea...ac58 bitcoin-0.17.0-osx.dmg
98ef...785e bitcoin-0.17.0.tar.gz
1f40...8ee7 bitcoin-0.17.0-win32-setup.exe
402f...730d bitcoin-0.17.0-win32.zip
b37f...0b1a bitcoin-0.17.0-win64-setup.exe
d631...0799 bitcoin-0.17.0-win64.zip
9d6b...5a4f bitcoin-0.17.0-x86_64-linux-gnu.tar.gz
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.11 (GNU/Linux)

iQIcBAEBCAAGBQJbtIOFAAoJEJDIAZ42wulk5aQP/0tQp+EwFQPtSJgtmjYucw8L
SskGHj76SviCBSfCJ0LKjBdnQ4nbrIBsSuw0oKYLVN6OIFIp6hvNSfxin1S8bipo
hCLX8xB0FuG4jVFHAqo8PKmF1XeB7ulfOkYg+qF3VR/qpkrjzQJ6S/nnrgc4bZu+
lXzyUBH+NNqqlMeTRzYW92g0zGMexig/ZEMqigMckTiFDrTUGkQjJGzwlIy73fXI
LZ/KtZYDUw82roZINXlp4oNHDQb8qT5R1L7ACvqmWixbq49Yqgt+MAL1NG5hvCSW
jiVX4fasHUJLlvVbmCH2L42Z+W24VCWYiy691XkZ2D0+bmllz0APMSPtgVEWDFEe
wcUeLXbFGkMtN1EDCLctQ6/DxYk3EM2Ffxkw3o5ehTSD6LczqNC7wG+ysPCjkV1P
O4oT4AyRSm/sP/o4qxvx/cpiRcu1BQU5qgIJDO+sPmCKzPn7wEG7vBoZGOeybxCS
UUPEOSGan1Elc0Jv4/bvbJ0XLVJPVC0AHk1dDE9zg/0PXof9lcFzGffzFBI+WRT3
zf1rBPKqrmQ3hHpybg34WCVmsvG94Zodp/hiJ3mGsxjqrOhCJO3PByk/F5LOyHtP
wjWPoicI2pRin2Xl/YTVAyeqex519XAnYCSDEXRpe+W4BdzFoOJwm5S6eW8Q+wkN
UtaRwoYjFfUsohMZ3Lbt
=H8c2
-----END PGP SIGNATURE-----
```

В подписанном сообщении в верхней части файла перечислено несколько файлов с соответствующими им хешами SHA256. Перечисленные файлы — это установочные пакеты для всех операционных систем и аппаратных архитектур, поддерживаемых Bitcoin Core. В нижней части файла находится подпись, удостоверяющая верхнюю часть. Подпись подтверждает все сообщение и, следовательно, все перечисленные хеши и имена файлов.

Проверка хеша загруженного файла

В этом примере мы загрузили файл `bitcoin-0.17.0-x86_64-linux-gnu.tar.gz`, поэтому его хеш SHA256 должен соответствовать хешу `9d6b...5a4f`. Давайте проверим:

```
$ sha256sum bitcoin-0.17.0-x86_64-linux-gnu.tar.gz
9d6b472dc2aceedb1a974b93a3003a81b7e0265963bd2aa0acdcb1759
↳8215a4f bitcoin-0.17.0-x86_64-linux-gnu.tar.gz
```

Эта команда вычислит хеш SHA256 загруженного файла. Он действительно соответствует хешу в файле SHA256SUMS.asc. Если у вас они не совпадают, значит, что-то пошло не так и вам следует прекратить установку и выяснить причину.

Получение открытого ключа Bitcoin Core

Чтобы убедиться, что подпись в файле SHA256SUMS.asc была создана с использованием закрытого ключа Bitcoin Core, нужно получить соответствующий открытый ключ. Как отмечалось в разделе «Шаг ❶ — запуск программного обеспечения», сначала нужно получить действительный отпечаток ключа Bitcoin Core, а затем загрузить сам ключ из любого источника.

Отпечаток ключа можно получить, например:

- на официальном веб-сайте команды Bitcoin Core https://bitcoincore.org;
- в книге «Грокаем технологию Битокин»;
- у друзей.

Сначала получим отпечаток открытого ключа на веб-сайте команды Bitcoin Core. Он находится на странице загрузки:

```
01EA5486DE18A882D4C2684590C8019E36C2E964
```

Теперь обратимся к книге «Грокаем технологию Биткоин» и проверим, совпадает ли отпечаток, указанный в ней, с отпечатком на сайте https://bitcoincore.org. В разделе «Шаг ❶ — запуск программного обеспечения» (глава 8) приводится отпечаток

```
01EA 5486 DE18 A882 D4C2 6845 90C8 019E 36C2 E964
```

Это тот же отпечаток (просто отформатирован немного иначе). Книга и веб-сайт https://bitcoincore.org подтверждают принадлежность этого ключа команде Bitcoin Core. Но не будем останавливаться на достигнутом и позвоним другу, которому доверяем, и попросим его прочитать отпечаток ключа:

Вы: «Привет, Донна! Прочитай мне отпечаток текущего открытого ключа Bitcoin Core».

Донна: «Привет! Я проверяла ключ пару месяцев назад, и тогда ему соответствовал отпечаток `01EA 5486 DE18 A882 D4C2 6845 90C8 019E 36C2 E964`».

Вы: «Спасибо, он совпадает с моим. Пока!»

Донна: «Всегда пожалуйста. Пока!»

Слова Донны еще больше укрепляют наше доверие к этому ключу. Похоже, мы собрали достаточно доказательств, что это действительно правильный ключ.

Теперь загрузим сам ключ. Для этого можно использовать инструмент с названием `gpg`, которое означает GnuPG, что в свою очередь означает Gnu Privacy Guard. Эта программа соответствует стандарту OpenPGP (Pretty Good Privacy). Этот стандарт определяет порядок обмена ключами, шифрования и создания цифровых подписей в интерактивном режиме.

Программное обеспечение GnuPG по умолчанию доступно на большинстве компьютеров с ОС Linux. Чтобы загрузить открытый ключ с определенным отпечатком, нужно выполнить следующую команду `gpg`:

```
$ gpg --recv-keys 01EA5486DE18A882D4C2684590C8019E36C2E964
gpg: key 90C8019E36C2E964: public key "Wladimir J. van der Laan (Bitcoin
↳Core binary release signing key) <laanwj@gmail.com>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1
```

В зависимости от версии `gpg`, вывод может немного отличаться от показанного выше. Эта команда скачает открытый ключ с любого доступного сервера ключей и проверит его соответствие указанному вами отпечатку. Владельцем этого ключа является «Wladimir J. van der Laan (Bitcoin Core binary release signing key)».

Предыдущая команда загрузит ключ и добавит его в наш список известных ключей. Но в выводе команды можно увидеть фразу «no ultimately trusted keys found» (доверенных ключей не найдено). Это означает, что данный ключ не подписан никаким другим ключом, которому мы доверяем. Мы только импортировали ключ. В gpg одни ключи могут подписывать другие ключи и тем самым подтверждать подлинность подписанного ключа.

Подписание открытого ключа на нашем компьютере как доверенного

Мы убедились, что ключ действительно принадлежит команде Bitcoin Core, и установили его в свою систему с помощью gpg.

Теперь подпишете его своим закрытым ключом, чтобы запомнить этот ключ как доверенный. Команда Bitcoin Core наверняка выпустит новые версии Bitcoin Core в будущем, и если GnuPG запомнит этот открытый ключ как доверенный, нам не придется снова выполнять шаги проверки ключа при обновлении.

Вот как выглядит эта процедура:

1. Создать свой ключ.

2. Подписать открытый ключ Bitcoin Core своим закрытым ключом.

Создать свой ключ в GnuPG можно, выполнив следующую команду:

```
$ gpg --gen-key
gpg (GnuPG) 2.1.18; Copyright (C) 2017 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation
↳dialog.
GnuPG needs to construct a user ID to identify your key.
```

GnuPG предложит ввести имя и адрес электронной почты. Ответим на эти вопросы; они будут использоваться для идентификации ключа:

```
Real name: Kalle Rosenbaum
Email address: kalle@example.com
You selected this USER-ID:
    "Kalle Rosenbaum <kalle@example.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit?
```

Продолжим, нажав клавишу O (заглавная латинская буква «O»). Затем нам нужно выбрать пароль для шифрования закрытого ключа. Выберите пароль и запомните его.

Чтобы сгенерировать ключ, может понадобиться некоторое время, потому что создание хороших случайных чисел для ключа — не самая быстрая процедура. По завершении команда выведет следующее:

```
public and secret key created and signed.

pub   rsa2048 2018-04-27 [SC] [expires: 2020-04-26]
```

```
      B8C0D19BB7E17E5CEC6D69D487C0AC3FEDA7E796
      B8C0D19BB7E17E5CEC6D69D487C0AC3FEDA7E796
uid                   Kalle Rosenbaum <kalle@example.com>
sub   rsa2048 2018-04-27 [E] [expires: 2020-04-26]
```

Теперь у нас есть свой ключ, который можно использовать для подписания доверенных ключей. Давайте подпишем ключ команды Bitcoin Core:

```
$ gpg --sign-key 01EA5486DE18A882D4C2684590C8019E36C2E964
pub   rsa4096/90C8019E36C2E964
      created: 2015-06-24 expires: 2019-02-14 usage: SC
      trust: unknown validity: unknown
[ unknown] (1). Wladimir J. van der Laan (Bitcoin Core binary release
↳signing key) <laanwj@gmail.com>
pub   rsa4096/90C8019E36C2E964
      created: 2015-06-24 expires: 2019-02-14 usage: SC
      trust: unknown validity: unknown
Primary key fingerprint: 01EA 5486 DE18 A882 D4C2 6845 90C8 019E 36C2
↳E964
      Wladimir J. van der Laan (Bitcoin Core binary release signing key)
↳<laanwj@gmail.com>
This key is due to expire on 2019-02-14.
Are you sure that you want to sign this key with your
key "Kalle Rosenbaum <kalle@example.com>" (8DC7D3846BA6AB5E)

Really sign? (y/N)
```

Нажмем клавишу y. Далее будет предложено ввести пароль личного ключа. Введем его и нажмем Enter. Теперь ключ Bitcoin Core будет считаться доверенным. Это упростит процесс обновления узла в будущем.

Посмотрим на вновь подписанный ключ:

```
$ gpg --list-keys 01EA5486DE18A882D4C2684590C8019E36C2E964
pub   rsa4096 2015-06-24 [SC] [expires: 2019-02-14]
      01EA5486DE18A882D4C2684590C8019E36C2E964
uid          [ full ] Wladimir J. van der Laan (Bitcoin Core binary release
↳signing key) <laanwj@gmail.com>
```

Найдите слово `full` в квадратных скобках. Оно означает, что gpg, как и вы, теперь полностью доверяет этому ключу.

Проверка подписи

Настал момент проверить подпись файла SHA256SUMS.asc:

```
$ gpg --verify SHA256SUMS.asc
gpg: Signature made Wed 03 Oct 2018 10:53:25 AM CEST
gpg:                using RSA key 90C8019E36C2E964
gpg: Good signature from "Wladimir J. van der Laan (Bitcoin Core binary
↳release signing key) <laanwj@gmail.com>" [full]
```

Вывод команды утверждает, что подпись верная (`Good`) и подписана ключом, которому мы полностью (`[full]`) доверяем.

Подведем итог сделанному:

1. Мы загрузили программное обеспечение Bitcoin Core и файл подписи.
2. Проверили совпадение хеша файла .tar.gz с хешем, указанным в файле SHA256SUMS.asc.
3. Загрузили открытый ключ и убедились в его принадлежности команде Bitcoin Core.
4. Подписали этот ключ своим закрытым ключом, чтобы GnuPG и мы запомнили, что ключ Bitcoin Core является действительным.
5. Проверили подпись файла SHA256SUMS.asc.

Когда позднее у нас появится желание обновить программное обеспечение, мы сможем пропустить некоторые из этих шагов. В этом случае процесс будет выглядеть так:

1. Загрузить Bitcoin Core и файл подписи.
2. Проверить совпадение хеша файла .tar.gz с хешем, указанным в файле SHA256SUMS.asc.
3. Проверить подпись файла SHA256SUMS.asc.

Распаковка и запуск

Распакуем программное обеспечение:

```
$ tar -zxvf bitcoin-0.17.0-x86_64-linux-gnu.tar.gz
```

Эта команда создаст каталог bitcoin-0.17.0. Перейдем в каталог bitcoin-0.17.0/bin и посмотрим, что в нем имеется:

```
$ cd bitcoin-0.17.0/bin
$ ls
bitcoin-cli bitcoind bitcoin-qt bitcoin-tx test_bitcoin
```

В каталоге имеется несколько выполняемых программ:

- `bitcoin-cli` — это программа, которую можно использовать для получения информации о текущем узле, а также для управления встроенным кошельком, который распространяется в составе Bitcoin Core;

- `bitcoind` — эта программа используется, когда желательно запустить узел в фоновом режиме без графического интерфейса пользователя;
- `bitcoin-qt` — программа, запускающая полный узел с графическим интерфейсом; обычно она используется, когда нужен доступ к встроенному кошельку;
- `bitcoin-tx` — небольшая утилита для создания и изменения транзакций Биткоин;
- `test_bitcoin` — позволяет выполнить комплект тестов.

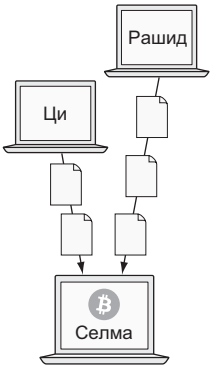
В этом примере мы запустим программу `bitcoind`. Ее имя означает «Bitcoin daemon» (демон Биткоин). В системах UNIX, таких как Linux, слово *демон* используется для обозначения программ, работающих в фоновом режиме.

Запустим демон Биткоин в фоновом режиме и посмотрим, что из этого получится:

```
$ ./bitcoind -daemon
Bitcoin server starting
```

Эта команда запускает полный узел. Он автоматически начнет подключаться к соседним узлам и загрузит блокчейн.

Начальная загрузка блокчейна



Этот процесс займет какое-то время. В зависимости от скорости подключения к Интернету, быстродействия процессора и диска на эту процедуру может уйти от нескольких дней до нескольких часов.

Чтобы проверить ход загрузки, можно воспользоваться программой `bitcoin-cli`, как показано ниже:

```
$ ./bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 207546,
  "headers": 549398,
  "bestblockhash":
↳"00000000000003a6a5f2f360f02a3b8e4c214d27bd8e079a70f5fb630a0817c5",
  "difficulty": 3304356.392990344,
  "mediantime": 1352672365,
  "verificationprogress": 0.0249296506976196,
  "initialblockdownload": true,
  "chainwork":
↳"000000000000000000000000000000000000000000000202ad90c17ec6ea33c",
  "size_on_disk": 11945130882,
```

```
  "pruned": false,
  "softforks": [
    {
      "id": "bip34",
      "version": 2,
      "reject": {
        "status": false
      }
    },
    {
      "id": "bip66",
      "version": 3,
      "reject": {
        "status": false
      }
    },
    {
      "id": "bip65",
      "version": 4,
      "reject": {
        "status": false
      }
    }
  ],
  "bip9_softforks": {
    "csv": {
      "status": "defined",
      "startTime": 1462060800,
      "timeout": 1493596800,
      "since": 0
    },
    "segwit": {
      "status": "defined",
      "startTime": 1479168000,
      "timeout": 1510704000,
      "since": 0
    }
  },
  "warnings": ""
}
```


Эта команда выводит большой объем информации о блокчейне. Обратите внимание, что в данном случае будут загружаться и проверяться блоки до высоты 207546. Bitcoin Core сначала загрузит заголовки блоков, чтобы проверить доказательство работы, и только потом начнет загружать полные блоки. В этом примере узел загрузил заголовки до высоты 54398, то есть все заголовки, существующие на этот момент. Также обратите внимание на интересное поле `initialblockdownload`, которое будет содержать значение `true`, пока начальная загрузка блоков не будет завершена.

Пусть этот демон продолжает работу. Мы еще вернемся к нему в приложении А, где я расскажу, как с помощью `bitcoin-cli` можно исследовать блокчейн и как пользоваться встроенным кошельком.

Остановить узел можно командой

```
$ ./bitcoin-cli stop
```

Вы в любой момент сможете запустить узел снова, и он продолжит работу с того места, где остановился.

Повторение

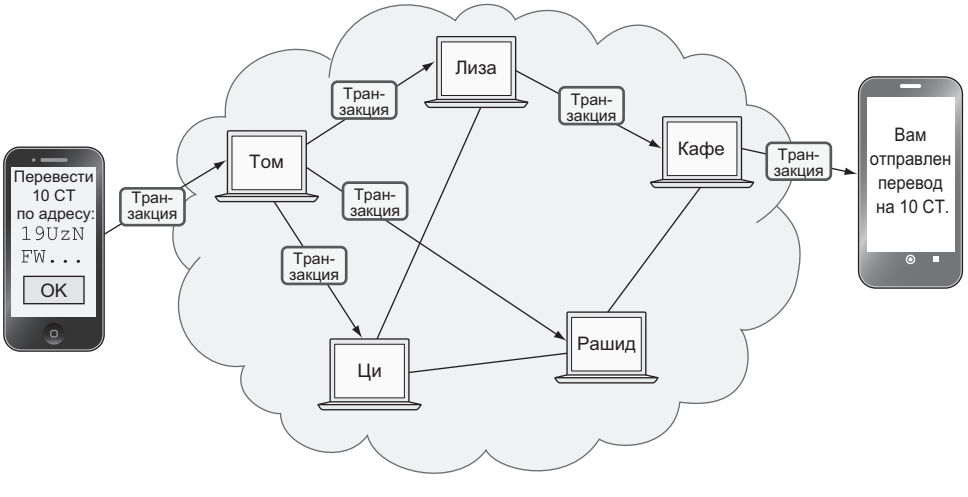
Мы заменили последнюю точку централизованной власти, общую папку, одноранговой сетью. Полные узлы в этой сети напрямую обмениваются данными друг с другом. Каждый узел связан с несколькими (потенциально сотнями) другими узлами. Это затрудняет препятствование распространению блоков и транзакций в сети.

Эта глава была поделена на две большие части:

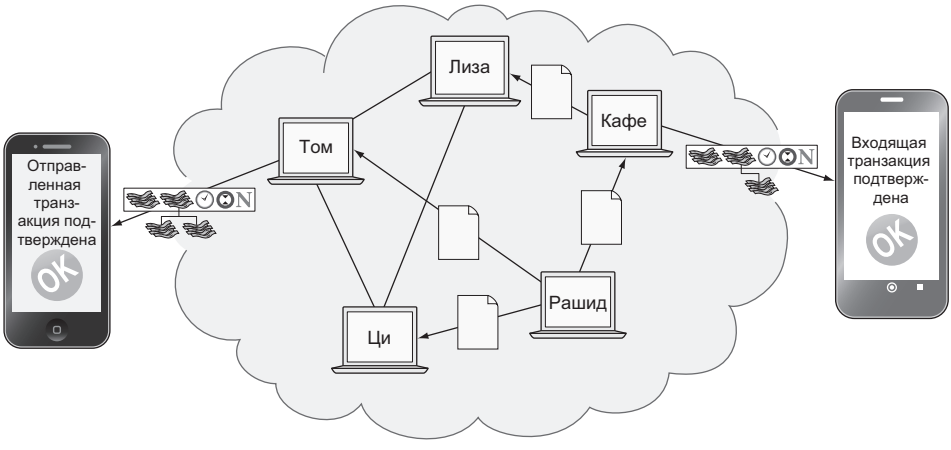
- как транзакции и блоки передаются по сети;
- как новые узлы включаются в сеть.

Часть 1 — передача транзакций

В первой части мы увидели, как перемещается транзакция через систему. Все началось с покупки булочки Джоном. Его транзакция распространялась по одноранговой сети и наконец попала в кошелек кафе.



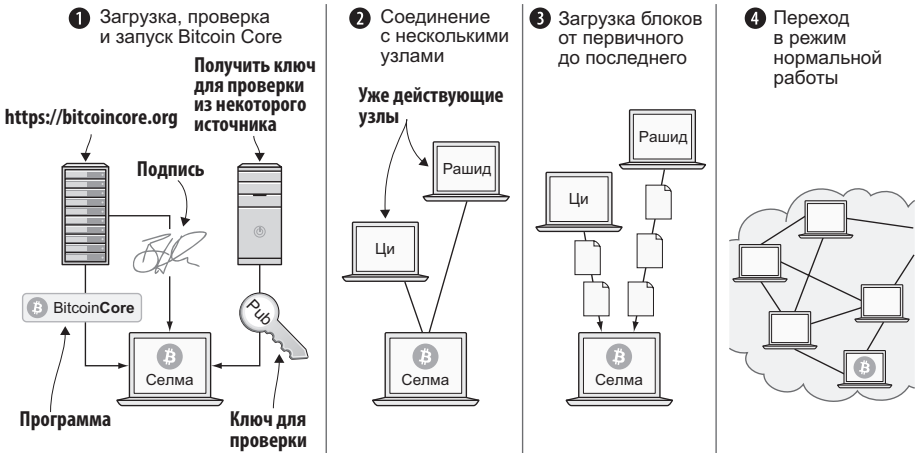
В кафе почти сразу заметили входящую, но пока не подтвержденную транзакцию. Следующий этап — майнинг блока. Рашиду повезло больше других, и он создал следующий блок, содержащий транзакцию Джона.



Рашид отослал блок своим соседям, которые передадут его своим соседям и так далее, пока блок не распространится по всей сети. Попутно блок будет отправляться на легкие кошельки. Эти кошельки будут запрашивать сообщения `merkleblock` у полного узла, и поэтому им не придется загружать полный блок.

Часть 2 — присоединение к сети

Включение в работу нового узла выполняется в четыре этапа:



❶ Загрузка и проверка программного обеспечения, например Bitcoin Core, и его запуск.

❷ Соединение с другими узлами.

❸ Загрузка существующих блоков.

❹ Переход в режим нормальной работы.

Изменения в системе

Таблица понятий, отражающая соответствие понятий между системами жетонов на булочки и Биткоин, сократилась до одной записи (табл. 8.2).

Таблица 8.2. Общую папку заменила одноранговая сеть

Жетоны на булочки	Биткоин	Где описывается
1 жетон на булочки	1 биткоин	Глава 2

Учитывая, что между системами жетонов на булочки и Биткоин не осталось никаких технических различий, мы оставим жетоны в прошлом и отныне будем работать только с Биткоин.

Это заключительный релиз системы жетонов на булочки. Мир покорила другая, гораздо более широко используемая система Биткоин, и мы решили прекратить разработку системы жетонов на булочки. А теперь просто порадуемся выходу новой версии (табл. 8.3).

Таблица 8.3. Примечания к релизу, жетоны на булочки 7.0

Версия	Особенность	Как реализована
НОВАЯ 8.0	Противодействие цензуре, на этот раз настоящее	Общую папку заменила одноранговая сеть
	Широковещательная рассылка транзакций	Рассылка транзакций майнерам и другим участникам посредством одноранговой сети
7.0	Противодействие цензуре	В работу по поиску доказательства работы включается несколько майнеров
	Любой может включиться в майнинг	Автоматическое регулирование сложности

Таблица 8.3 (*окончание*)

Версия	Особенность	Как реализована
6.0	Лиза лишена возможности удалять транзакции	Подписи блоков в блокчейне
	Полная проверка узлами	Узлы загружают и проверяют весь блокчейн
	Легкие кошельки экономят трафик	Нечеткие фильтры и доказательство Меркла

Упражнения

Для разминки

8.1. Почему общая папка — не лучшее решение?

8.2. Что подразумевается под *пересылкой* транзакции или блока?

8.3. Для чего используется сообщение `inv`?

8.4. Как полный узел определяет, какие транзакции посылать легким кошелькам?

8.5. Как полный узел уведомляет легкий кошелек о появлении входящей транзакции, ожидающей подтверждения?

8.6. Блоки не посылаются легким кошелькам. Какая часть блока всегда посылается в кошелек?

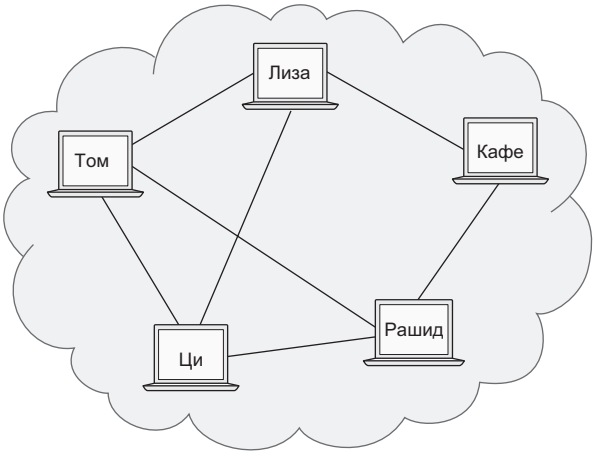
8.7. Почему легкий кошелек в кафе посылает очень большой фильтр Блума своему доверенному узлу?

8.8. Что должен сделать человек, заботящийся о безопасности, после загрузки программного обеспечения Bitcoin Core, но до его запуска?

8.9. Из каких источников вновь созданный узел может узнать адреса соседей?

8.10. Как полный узел узнает о появлении новых блоков по завершении синхронизации?

8.11. Одноранговая сеть Биткоин состоит из следующих узлов:



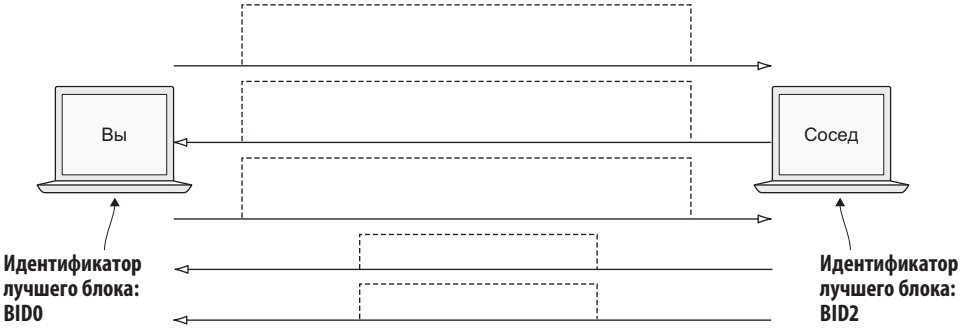
Владельцев каких узлов нужно подкупить или запугать, чтобы Лиза не смогла получать никаких новых блоков, кроме тех, что создает сама?

Придется пораскинуть мозгами

8.12. Предположим, что Ци только что получила две транзакции с идентификаторами транзакций $TXID_1$ и $TXID_2$. Теперь она хочет сообщить о них Рашиду, но не знает, получал ли Рашид эти транзакции от других. Как она поступит?

8.13. Предположим, у вас есть полный узел и в какой-то момент пропала электроэнергия. Когда через 18 минут подача электроэнергии восстановилась, вы снова запускаете свой узел и обнаруживаете, что за это время было создано два блока, B_1 и B_2. Последний блок, известный вам, — B_0. Что сделает ваш узел после повторного подключения к сети? Для простоты предположим, что во время синхронизации не найдено никаких новых блоков и что у вас есть только один сосед. Используйте следующую таблицу типов сообщений и заполните карточку на рисунке ниже.

Тип	Данные	Назначение
`block`	Полный блок	Отправка блока соседу
`getheaders`	Идентификатор блока	Запросить у соседа порцию заголовков блоков, следующих за указанным
`getdata`	Идентификаторы транзакций или блоков	Запросить данные у соседа
`headers`	Список заголовков	Отправка списка заголовков соседу



Итоги

- Одноранговая сеть устраняет возможность цензуры блоков.
- Узел подключается к нескольким соседям, чтобы уменьшить уязвимость к попыткам скрыть информацию.
- Сетевой протокол Биткоин — это «язык», на котором общаются узлы.
- Транзакции распространяются в одноранговой сети Биткоин с целью как можно быстрее доставить их всем майнерам и получателям денег.
- Новые узлы синхронизируются с сетью Биткоин для достижения актуального состояния, соответствующего состоянию других узлов. Это может занимать часы или дни.
- Узлам необязательно оставаться в сети круглые сутки и без выходных. Узел можно остановить в любой момент, и после включения он автоматически синхронизируется до актуального состояния.
- Проверку подписей в старых блоках можно пропустить, чтобы ускорить начальную синхронизацию. Такой способ можно использовать, если известно, что конкретный блок является действительным.

9 И снова о транзакциях

Эта глава охватывает следующие темы:

- ✓ временная блокировка биткоинов;
- ✓ перевод монет между блокчейнами;
- ✓ присоединение произвольных данных к транзакции;
- ✓ увеличение комиссии за ожидающую транзакцию.

Мы осилили основные главы книги, где познакомились с базовыми понятиями системы Биткоин. В этой главе мы углубимся в исследование возможностей, предлагаемых транзакциями.

Начнем с изучения временной блокировки. *Временная блокировка* (time lock) обеспечивает недействительность транзакции до определенного момента времени и откладывает ее подтверждение. Кроме того, выход транзакции можно запрограммировать так, чтобы предотвратить его расходование, пока действует временная блокировка. Это полезно для цифровых контрактов, таких как атомарный обмен, или своп (atomic swap), о которых рассказывается далее.

Иногда вместе с транзакцией нужно сохранить в блокчейне небольшой объем данных. Например, производителю автомобилей нужно отследить передачу права собственности на автомобиль: он вводит номер шасси в транзакцию Биткоин и фактически создает ключ в блокчейне. Текущий владелец может затем передать право собственности на автомобиль, отправив этот ключ новому владельцу.

Из раздела «Другие криптовалюты» главы 1 мы знаем, что в мире существует несколько альтернативных криптовалют. Иногда монеты этих криптовалют можно обменять на биткоины. Самый очевидный способ сделать это — воспользоваться услугами биржи для продажи биткоинов и покупки монет другой криптовалюты. Но есть и более децентрализованные способы сделать это. Операции *атомарного обмена* (atomic swap) позволяют напрямую обменивать биткоины на монеты другой криптовалюты, без привлечения третьей доверенной стороны, например биржи.

Если вы заплатите слишком маленькую комиссию за транзакцию, майнеры могут отказаться подтверждать ее в течение довольно долгого времени. В этой ситуации полезно заменить такую «зависшую» транзакцию другой, содержащей большую сумму комиссионных. Этот прием называют *комиссионным сбором* (fee-bumping).

В заключение мы рассмотрим некоторые тонкости подписей. Подписи можно создавать по-разному, в зависимости от варианта использования. Можно определить, что именно удостоверяет подпись, то есть определить, как алгоритм подписания должен хешировать транзакцию.

Временная блокировка транзакций

Сразу после создания и подписания транзакция действительна и готова для включения в любой будущий блок. Ее можно сразу же разослать майнерам. Это обычный случай.

Но в некоторых случаях может понадобиться подписать транзакцию с гарантией того, что она не будет включена в блок, по крайней мере, по истечении года.

Предположим, у вас есть 100 биткоинов и вы хотите завещать их своей дочери, отправив на ее адрес $@_D$, но с условием, что она сможет получить их только после вашей смерти. Вы можете создать транзакцию с временной блокировкой (рис. 9.1).

БЕЗ КОМИССИИ?

В целях упрощения большинство примеров в этой главе не включают комиссионных отчислений.

Отличительной особенностью этой транзакции являются порядковые номера входов и время блокировки. Я уже упоминал порядковые номера в главе 5. Они используются для включения времени блокировки: если какой-то вход имеет порядковый номер меньше `ffffffff` — например, `fffffffe`, —

время блокировки в транзакции будет считаться действительным. Если все порядковые номера равны `ffffffff`, время блокировки не будет иметь никакого эффекта.

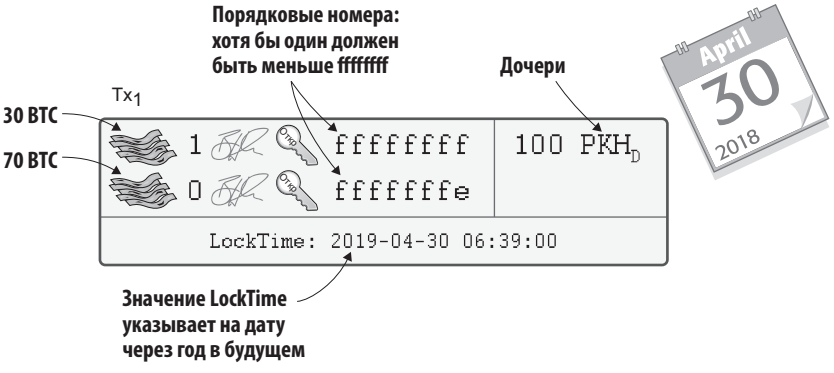


Рис. 9.1. Перевод денег дочери, который станет действительным с 30 апреля 2019 года

Далее вы посылаете транзакцию Tx_1 своей дочери. В данный момент транзакция недействительна; ваша дочь может сохранить ее на своем компьютере и распечатать резервную копию, чтобы сохранить в другом месте. Эту транзакцию пока нельзя разослать; ни один полный узел не примет блок с этой транзакцией. Транзакция станет действительной утром 30 апреля 2019 года. Если вы умрете до этого времени, вашей дочери придется дождаться истечения времени блокировки, а затем потребовать деньги, отправив в сеть транзакцию, которая к этому моменту станет действительной.

ПОРЯДКОВЫЕ НОМЕРА

Входы всегда включают порядковые номера, но раньше я не показывал их, потому что для того, как мы использовали транзакции, они не имели никакого значения.

Если вы не умерли до этой даты, то сможете отменить транзакцию с временной блокировкой, чтобы дочь не смогла получить деньги после наступления заданной даты.

С этой целью вы создаете, но пока не отправляете новую транзакцию Tx_2, которая повторно расходует тот же выход, что и Tx_1 (рис. 9.2). Затем создаете новую транзакцию Tx_3 с временной блокировкой еще на один год. Когда ваша дочь сохранит эту новую транзакцию в безопасном месте, вы посылаете Tx_2.

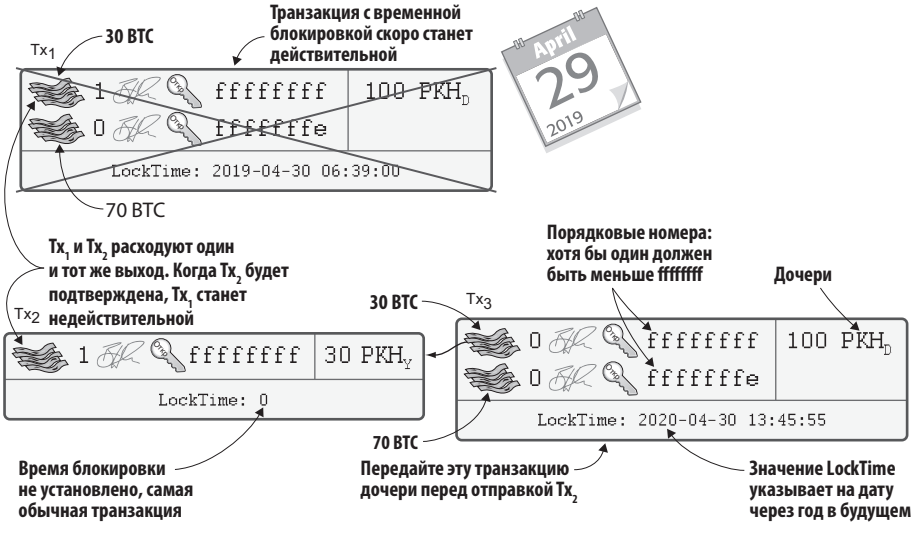


Рис. 9.2. Отмена транзакции Tx_1 повторным расходованием того же выхода, который расходует Tx_1, и создание новой транзакции с блокировкой по времени

Вы должны:

1. Создать и подписать транзакцию Tx_2, расходующую хотя бы один из выходов, расходуемых транзакцией Tx_1. Tx_2 — это обычная транзакция без временной блокировки. Но пока не отправляйте эту транзакцию.

2. Создайте новую транзакцию с временной блокировкой, Tx_3, расходующей все выходы, как если бы транзакция Tx_2 была подтверждена. Tx_3 включает временную блокировку еще на один год. Передайте ее своей дочери.

3. Отправьте транзакцию Tx_2. Когда Tx_2 будет включена в блок, Tx_1 станет недействительной, потому что один из входов Tx_1 уже израсходован транзакцией Tx_2.

Обратите внимание: порядок действий играет важную роль. Если Tx_2 послать *перед* передачей Tx_3 дочери, есть шанс, что вы умрете раньше, чем успеете передать ей транзакцию Tx_3. Тогда

ПЛАСТИЧНОСТЬ ТРАНЗАКЦИЙ

Здесь есть одна проблема. Идентификатор транзакции Tx_2 *можно* изменить во время распространения и сделать Tx_3 недействительной. Эта проблема называется *пластичностью транзакции* (transaction malleability) и устраняется выделением подписей и сценариев в отдельную структуру *SegWit* (segregated witness), как описано в главе 10.

ваша дочь не сможет получить средства, потому что у нее не будет действительной транзакции, послав которую она сможет получить наследство. Tx_1 окажется аннулированной транзакцией Tx_2 в блокчейне, а Tx_3 так и не попала в руки вашей дочери.

Измерение времени

Время блокировки можно выразить двумя способами. Первый способ: указать время и дату, как в предыдущем примере. Второй: задать высоту блока.

Время блокировки

В первом примере окончание блокировки определялось датой и временем. В этом случае конечное время блокировки в транзакции должно оказаться старше *медианы прошедшего времени* (median time past). В главе 7 я отмечал, что время создания блока должно быть больше медианы времени создания последних 11 блоков, то есть *медианы прошедшего времени*. Медиана прошедшего времени также используется, чтобы определить, является ли действительной транзакция с временной блокировкой. Предположим, вы умерли 24 января 2019 года. Ваша безутешная дочь не сможет истребовать денег до 30 апреля 2019 года, как показывает рис. 9.3.

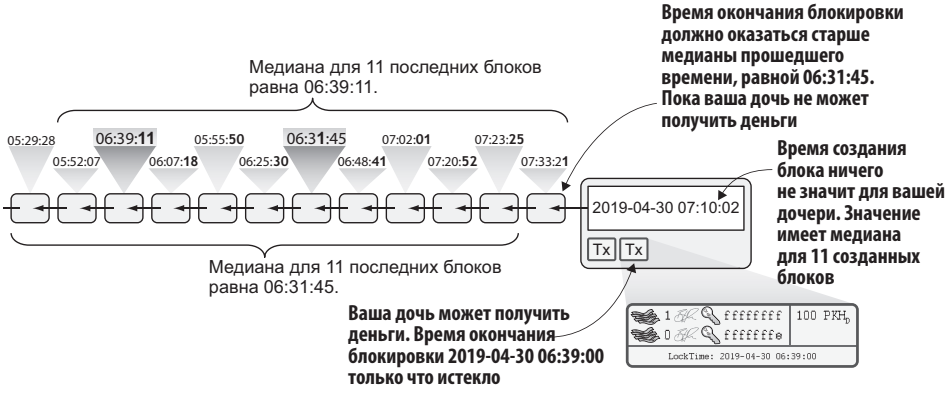


Рис. 9.3. Ваша дочь сможет претендовать на наследство, когда конечное время блокировки окажется старше медианы прошедшего времени

Транзакция вашей дочери не сможет попасть ни в один из блоков до последнего, изображенного на рис. 9.3. До этого блока медиана прошедшего времени оказывается старше времени окончания блокировки транзакции.

Ее транзакция просто не будет распространяться по сети Биткоин, пока не минует время блокировки. Узлы не желают хранить транзакции с временной блокировкой в своей памяти, потому что их драгоценной памяти можно найти лучшее применение, чем заполнение транзакциями, которые даже не действительны (пусть и временно). Ваша дочь может отправить транзакцию после истечения времени блокировки.

Высота блока

Временную блокировку также можно выразить с использованием высоты блока. Например, можно потребовать, чтобы транзакция оставалась недействительной до появления блока с высотой 571019. Это означает, что транзакция, изображенная на рис. 9.4, не будет подтверждена, пока не будет добыт блок 571019.

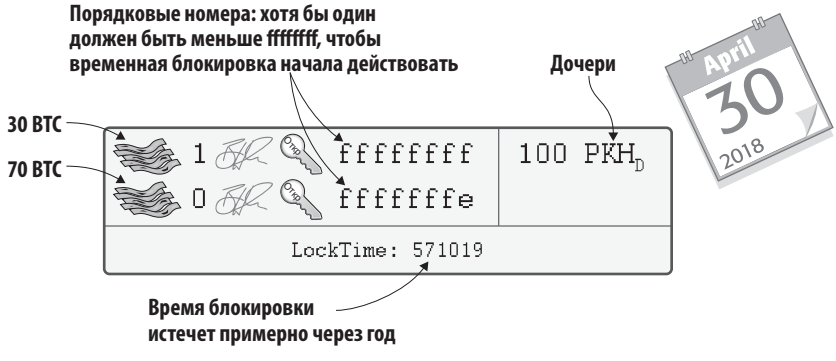


Рис. 9.4. Временная блокировка транзакции с использованием высоты блока. Эта транзакция впервые станет действительной в блоке с высотой 571020

BIP68

Это предложение по улучшению Биткоин (Bitcoin Improvement Proposal, BIP) описывает, как вход может требовать определенной удаленности во времени блоков с транзакциями, выходы которых расходуют этот вход. Это относится к транзакциям с версией не ниже 2.

Самый ранний блок, в который может быть включена транзакция, находится на высоте 571020. Трудно предсказать точно, когда этот блок будет добыт, но благодаря корректировкам сложности, которые удерживают среднее время

появления нового блока примерно равным 10 минутам, можно ожидать, что за год будет произведено 52 596 блоков.

Относительные временные блокировки

В предыдущем примере мы использовали абсолютные временные блокировки транзакций. Но точно так же можно блокировать входы транзакций, пока ее расходуемый выход не станет достаточно старым. Это называется *относительной временной блокировкой*. Такая блокировка осуществляется для каждого входа в отдельности (рис. 9.5).

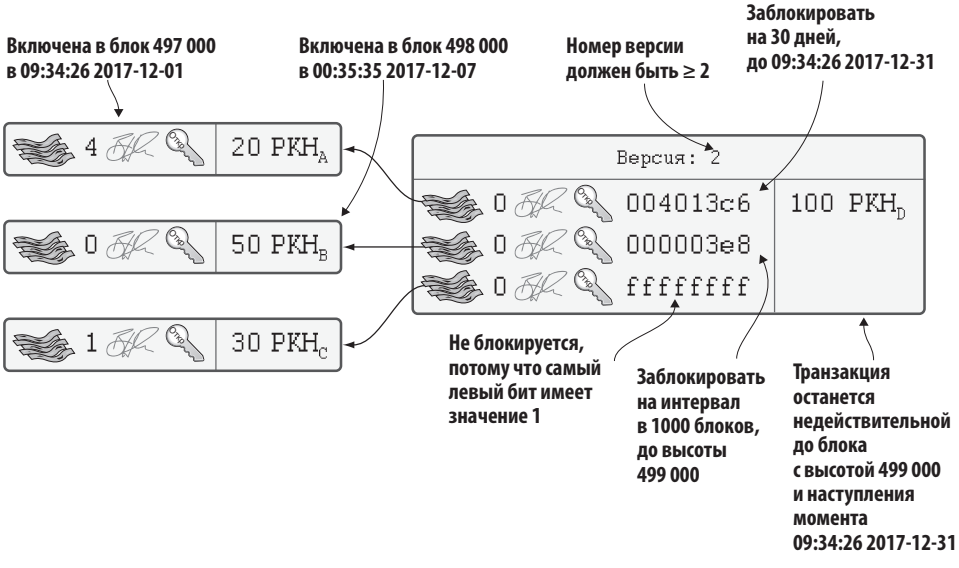


Рис. 9.5. Относительные временные блокировки выражаются либо числом блоков, либо числом единиц времени. Для этого используются порядковые номера во входах

Первый вход транзакции имеет порядковый номер `004013c6`. Это говорит о том, что транзакция недействительна, пока не пройдет 30 дней с момента подтверждения потраченного выхода (рис. 9.6).

Крайний левый бит в этом порядковом номере равен 0. Это означает, что относительная временная блокировка включена. Бит с индексом 9, если считать слева, равен 1. Это означает, что самые правые 16 бит должны интерпретироваться как «число 512-секундных интервалов». Шестнадцать

самых правых битов содержат значение `13c6`, которое равно десятичному значению 5062; то есть оно определяет 5062 512-секундных интервалов, или примерно 30 дней.

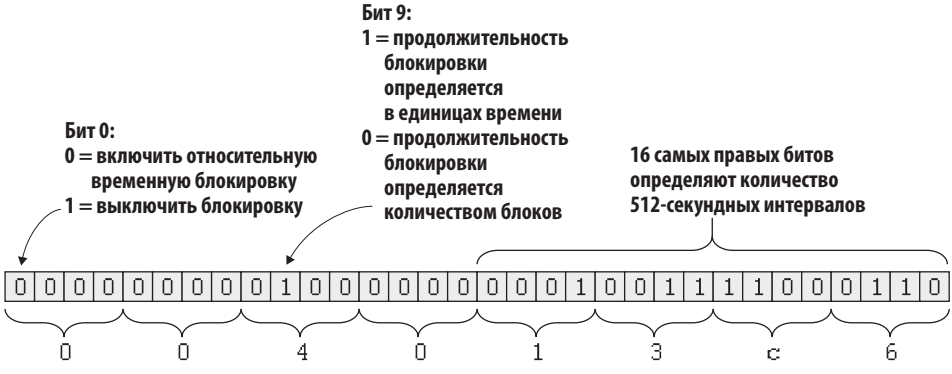


Рис. 9.6. Первый вход блокирует транзакцию, пока не пройдет 30 дней с момента подтверждения потраченного выхода

Второй вход имеет порядковый номер `000003e8` (рис. 9.7). Он означает, что транзакция недействительна, пока не будет найдена тысяча блоков после того, как был подтвержден расходуемый выход.

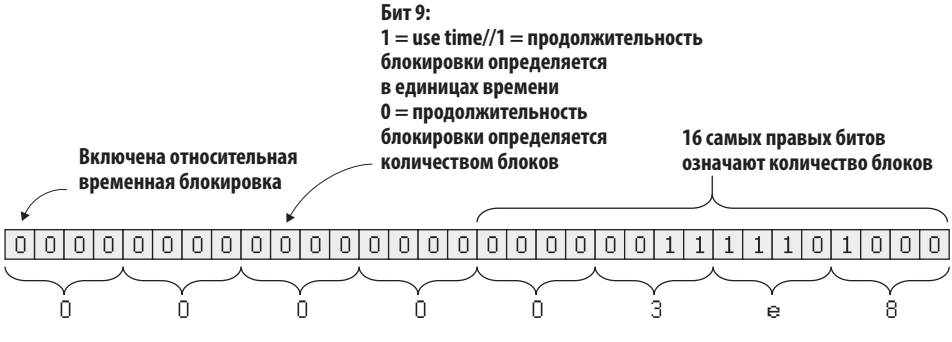


Рис. 9.7. Второй вход блокирует транзакцию, пока не будет найдено 1000 блоков после того, как был подтвержден расходуемый выход

Самый левый бит здесь тоже равен 0, то есть для этого входа включена относительная временная блокировка. Бит с индексом 9, если считать слева, равен 0. Это означает, что самые правые 16 бит должны интерпретироваться как количество блоков; `03e8` — это шестнадцатеричное представление десятичного числа 1000.

Чтобы относительные временные блокировки работали, версия транзакции должна быть не ниже 2. В транзакции с версией 1 порядковые номера будут определять не относительную, а абсолютную временную блокировку и порядок замены за плату, о которой я расскажу позже в разделе «Замена ожидающих транзакций».

Временная блокировка выходов

Временные блокировки не особенно полезны сами по себе. Единственное, что они позволяют, — создать транзакцию, которая рано или поздно может стать действительной.

Полезнее было бы сказать, например: «Деньги в этом выходе нельзя потратить до Нового года». Это пример *временной блокировки выхода*. Выход можно заблокировать по времени или высоте, указав абсолютное или относительное значение.

Временная блокировка выходов по абсолютному значению

Предположим, вы решили перевести своей дочери 1 BTC в подарок на 1 мая. Для этого можно создать транзакцию, как показано на рис. 9.8.

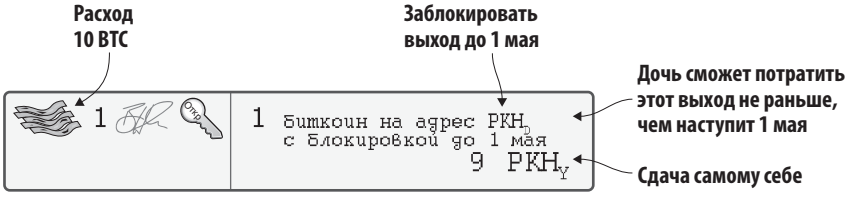


Рис. 9.8. Перевод денег в подарок вашей дочери. Она не сможет потратить их до 1 мая

BIP65

В этом предложении по улучшению описываются детали оператора сценариев `OP_CHECKLOCKTIMEVERIFY`, который реализует временную блокировку выхода по абсолютному значению.

Эту транзакцию можно сразу же отправить в сеть Биткоин и включить ее в блок. Наибольший интерес для нас представляет первый выход. Он говорит, что его нельзя потратить до 1 мая. Для любопытных покажу, как выглядит точный сценарий открытого ключа:

```
<1 may 2019 00:00:00> OP_CHECKLOCKTIMEVERIFY OP_DROP
OP_DUP OP_HASH160 <PKHD> OP_EQUALVERIFY OP_CHECKSIG
```

Этот сценарий гарантирует невозможность потратить соответствующий ему выход до наступления указанной даты, как показано на рис. 9.9.

OP_DROP?

Оператор `OP_CHECKLOCKTIMEVERIFY` требует, чтобы за ним следовал оператор `OP_DROP`. Это объясняется особенностями развертывания оператора в Биткоин. Подробнее об этом мы поговорим в главе 10, а пока просто игнорируйте эту особенность.

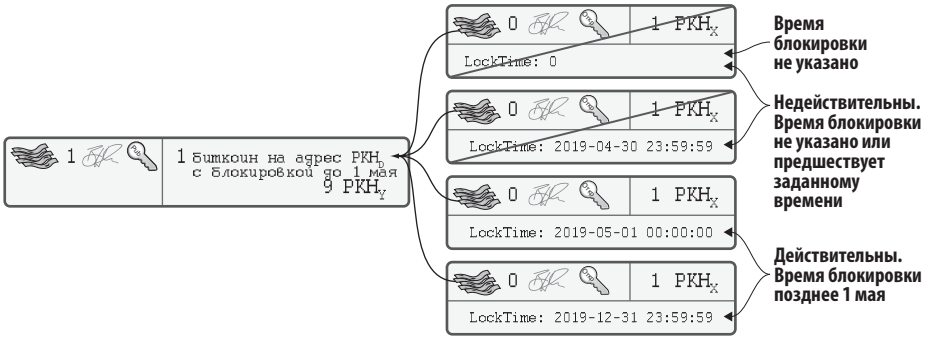


Рис. 9.9. Разные транзакции, расходующие средства, и их действительность

Первые две транзакции будут считаться недействительными, потому что указанное в них время блокировки не достаточно позднее. Первая вообще не определяет времени блокировки, что недопустимо, согласно сценарию открытого ключа. Вторая содержит время, но оно недостаточно позднее — еще остается 1 секунда до наступления 1 мая.

Третья транзакция — действительная, потому что указанное в ней время блокировки не меньше времени в сценарии открытого ключа, 00:00:00 2019-05-01. Транзакция, созданная 1 мая или позже, будет считаться действительной. Последняя транзакция, созданная в канун Нового года, прямо перед

фейерверком, тоже будет считаться действительной. Но обратите внимание, что из двух последних транзакций подтверждена будет только одна, потому что они тратят один и тот же выход.

Таким образом, ваша дочь сможет потратить выход полученной транзакции не раньше, чем наступит 1 мая.

Временная блокировка выходов по относительному значению

Временная блокировка выходов по относительному значению действует подобно блокировке по абсолютному значению с той лишь разницей, что время блокировки в этом случае определяет интервал, который должен *пройти* между блоком, содержащим расходуемый выход, и блоком, содержащим транзакцию, которая его расходует (рис. 9.10).

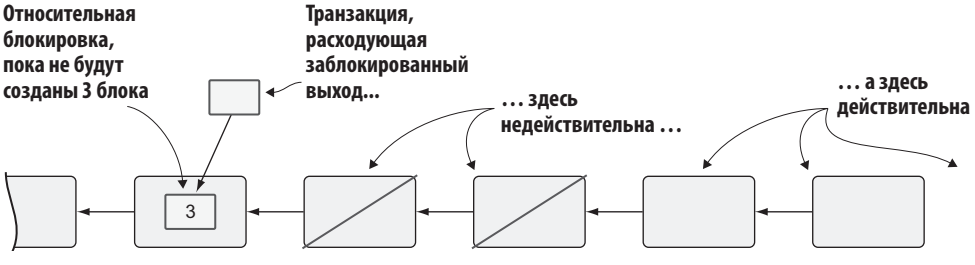


Рис. 9.10. Потратить выход с относительной временной блокировкой можно только после создания указанного числа блоков

Относительные временные блокировки чаще всего используются в *цифровых контрактах*. Цифровой контракт можно рассматривать как традиционный договор между сторонами, но действующий в соответствии с правилами сети Биткоин, а не национальных законодательств. Контракты выражаются в виде сценариев открытого ключа. Мы рассмотрим использование выходов с относительной временной блокировкой для атомарного обмена в следующем подразделе. Под атомарным обменом подразумевается обмен монетами разных криптовалют между двумя сторонами.

BIP112

В этом предложении по улучшению описываются детали оператора сценариев `OP_CHECKSEQUENCEVERIFY`, который реализует временную блокировку выхода по относительному значению.

Атомарный обмен

АТОМАРНЫЙ

В информатике словом *атомарный* обозначаются процессы, которые выполняются полностью, от начала до конца, или не выполняются вовсе. Атомарный обмен — это обмен, который либо совершается, либо обе стороны остаются при своих деньгах. Никакие другие исходы невозможны.

Под цифровыми контрактами часто подразумевается *атомарный обмен*, или *своп* (atomic swap), когда две стороны хотят обменяться монетами из разных блокчейнов.

Предположим, что Джон общается с Фатимой на публичном форуме в Интернете. Они не знают друг друга и не имеют причин доверять друг другу. Но оба хотят торговать.

Они решают обменять 2 BTC, принадлежащих Джону, на 100 монет Namecoin (NMC), принадлежащих Фатиме. Namecoin — это альтернативная криптовалюта, используемая в качестве децентрализованной системы имен, подобной DNS. Об альтернативных криптовалютах мы уже говорили в главе 1. В этом примере совершенно неважно, для чего используется Namecoin; отметим лишь, что это совершенно другая криптовалюта с собственным блокчейном, отличным от блокчейна Биткоин.

Вот как начинается диалог между Джоном и Фатимой:

Джон: Давай обменяем твои 100 NMC на мои 2 BTC? Мой открытый ключ Namecoin: `02381efd...88ca7f23`. Я создал секретное случайное число, имеющее хеш SHA256`H`. Но пока не скажу его тебе.

Фатима: Хорошо, давай обменяем! Мой открытый ключ в Биткоин: `02b0c907...df854ee8`.

Обозначим секретное число как `S`. Только Джон знает `S`, но он может передать Фатиме его хеш `H`. Теперь у обоих достаточно информации, чтобы начать.

Каждый из них создает свою транзакцию (рис. 9.11). Джон создает транзакцию Биткоин, которая тратит 2 BTC. Фатима создает транзакцию Namecoin, которая тратит 100 NMC. Пока они не отправляют свои транзакции.

Рис. 9.11. Джон и Фатима создают контрактные транзакции. Детали контракта определяются сценарием погашения выхода p2sh

Выход контрактной транзакции Джона можно потратить:

- Предоставив исходный образ хеша `H` и подпись Фатимы. Джон знает исходный образ — это его секретное число `S`, о котором говорилось выше, но Фатима не знает его.
- Предоставив подпись Джона спустя 48 часов.

Аналогично выход контрактной транзакции Фатимы можно потратить:

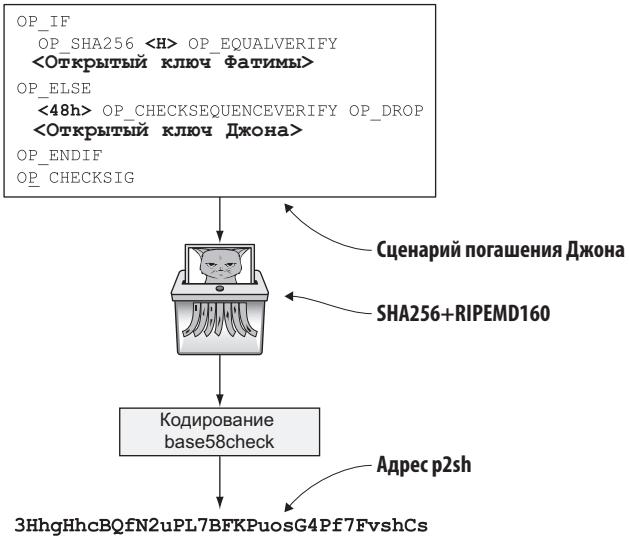
- Предоставив исходный образ хеша `H` и подпись Джона.
- Предоставив подпись Фатимы спустя 24 часа.

Оператор `OP_CHECKSEQUENCEVERIFY` устанавливает относительную временную блокировку. Он гарантирует, что Джон не сможет потратить выход своей контрактной транзакции, пока не пройдет 48 часов с момента ее подтверждения. Оператор в контрактной транзакции Фатимы гарантирует, что Фатима не сможет потратить ее выход, пока не пройдет 24 часа.

Фатима знает, что у Джона есть секретное число. Если Фатима отправит свою контрактную транзакцию прямо сейчас, Джон может забрать деньги

и не выполнить свою часть сделки. По этой причине она не посылает свою транзакцию, пока не увидит, что транзакция Джона благополучно подтверждена в блокчейне. Джон, напротив, может отправить свою контрактную транзакцию, не опасаясь остаться без денег, потому что Фатима не знает секретного числа S.

Джон посылает свою контрактную транзакцию в сеть. Напомню, что выход его контрактной транзакции — это платеж на хеш сценария (pay-to-script-hash, p2sh). Проще говоря, выход содержит адрес p2sh, который ничего не говорит о том, что это выход контрактной транзакции Джона. Чтобы идентифицировать контрактную транзакцию Джона в блокчейне Биткоин, Фатима должна создать тот же сценарий погашения, что и в контрактной транзакции Джона, и сгенерировать адрес p2sh, на который контрактная транзакция Джона переводит деньги. Затем она сможет найти этот адрес p2sh в блокчейне Биткоин.



Когда Фатима обнаружит, что транзакция Джона подтверждена, она отправит свою контрактную транзакцию. Джон дождется, пока транзакция Фатимы получит достаточное количество подтверждений в блокчейне Namecoin. Затем в два этапа выполнится фактический обмен. На рис. 9.12 показан первый этап.

Джон посылает свою обменную транзакцию. Обменная транзакция Джона тратит выход контрактной транзакции Фатимы, предоставляя S и его под-

пись. И снова обратите внимание, что Джон тратит выход p2sh. Это означает, что во время проверки подлинности сценария сначала будет вычислен хеш сценария погашения Джона, который он указал в сценарии подписи, и сравнен с хешем в сценарии открытого ключа. Затем будет выполнен фактический сценарий погашения.

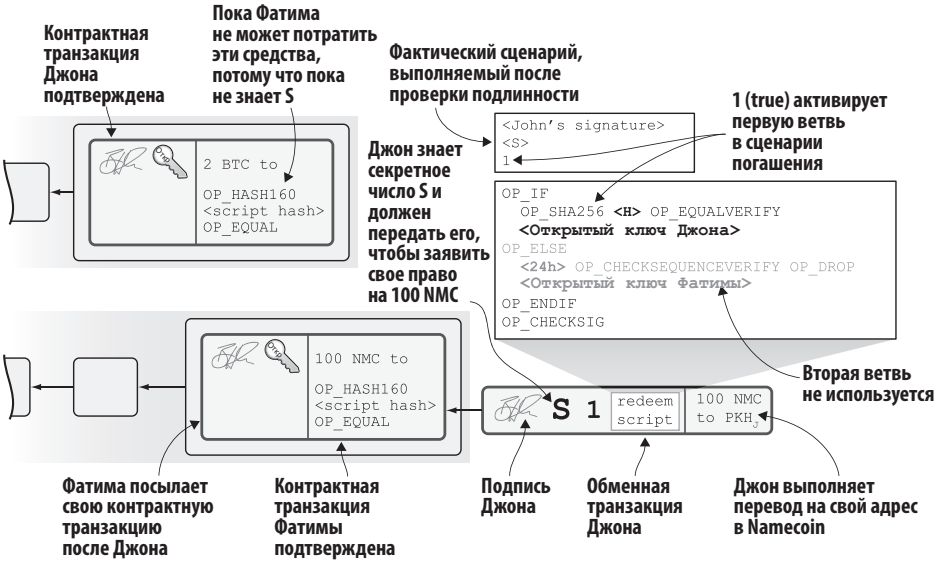


Рис. 9.12. Первый этап фактического обмена. Джон заявляет право на 100 NMC, принадлежащих Фатиме, передавая ей секретное число S

Я не буду подробно описывать, как работает программа, но отмечу, что когда запускается сценарий погашения, на вершине стека будет лежать число `1`. В Namecoin оно соответствует истинному значению (`true`), так же как и в Биткоин. В результате программа выполнит ту часть сценария, которая использует исходный образ и подпись Джона. Другая часть вообще не будет использована.

Сценарий оставит на вершине стека значение `true`, потому что Джон предоставил оба обязательных элемента в правильном порядке — свою подпись и исходный образ `S`. Он благополучно подтвердил свое право на 100 NMC.

Как только Фатима увидит обменную транзакцию Джона в сети Namecoin, она сможет создать свою обменную транзакцию для блокчейна Биткоин (рис. 9.13).

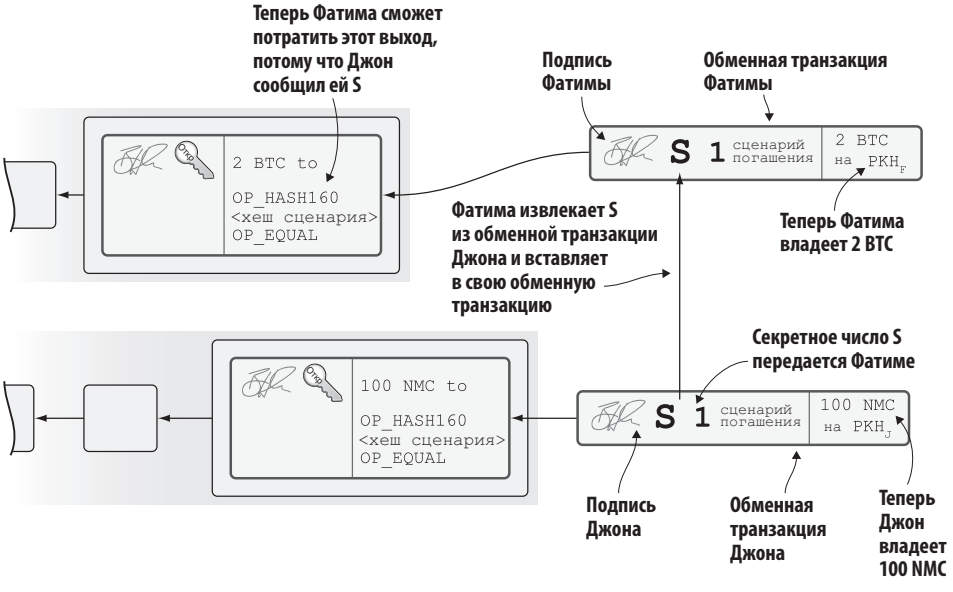


Рис. 9.13. Фатима завершает атомарный обмен, посылая свою обменную транзакцию в сеть Биткоин

Она извлекает исходный образ S из обменной транзакции Джона и вставляет в свою обменную транзакцию, которая переводит 2 BTC на хеш открытого ключа Фатимы PKH_F. После подтверждения обеих обменных транзакций атомарный обмен завершается. Суть всего этого процесса сводится к тому, что Джон отправляет 2 BTC Фатиме с условием, что та отправит ему 100 NMC, а Фатима отправляет 100 NMC Джону с условием, что тот отправит ей 2 BTC.

Неудачное завершение атомарного обмена

Последовательность событий, описанная в примере атомарного обмена выше, иллюстрирует случай, когда обе стороны, Джон и Фатима, играют по правилам. В действительности никто не был обязан использовать в выходах контрактных транзакций ветви с временными блокировками. В этом подразделе мы рассмотрим несколько случаев безуспешного завершения обмена:

- *Фатима не отправила свою контрактную транзакцию*. Вследствие этого Джон не сможет потратить выход контрактной транзакции Фатимы,

а значит, Фатима не получит S. Без S она не сможет потратить выход контрактной транзакции Джона. Единственный возможный исход: Джон вынужден будет подождать 48 часов, пока закончится относительная временная блокировка, после чего сможет вернуть свои деньги.

- *Джон не потратил контрактную транзакцию Фатимы в течение 24 часов*. Фатима может вернуть свои монеты, а Джону придется подождать еще 24 часа, после чего он тоже сможет вернуть свои монеты.
- *Джон потратил контрактную транзакцию Фатимы сразу после того, как прошло 24 часа, но до того, как Фатима успела забрать свои монеты*. К счастью, выход контрактной транзакции Джона имеет 48-часовой период блокировки, против 24 часов в выходе контрактной транзакции Фатимы, поэтому Джон не сможет вернуть свои монеты, пока не пройдут еще 24 часа. В течение этого времени Фатима может потребовать свои 2 BTC из выхода контрактной транзакции Джона, используя S и свою подпись.
- *Фатима попадает под автобус сразу после отправки своей контрактной транзакции*. Это плохо. Джон сможет забрать свои NMC из выхода контрактной транзакции Фатимы, а затем подождать 48 часов и вернуть свои BTC. Фатима оказывается в проигрыше.

В последнем случае можно утверждать, что обмен оказался неатомарным. В конце концов, обмен как таковой не состоялся — Джон забрал себе все монеты. Впрочем, это уже в большей степени философский вопрос. Но мы можем считать обмены атомарными при условии, что Фатима способна активно действовать. Однако это условие не предъявляется Джону. Все дело в том, кто создает секретное число S.

Сохранение данных в блокчейне Биткоин

Уже в первые дни существования Биткоин стало ясно, что люди не прочь сохранять в блокчейне вместе с транзакциями некоторые данные, не имеющие никакого отношения к самой системе Биткоин. Например, взгляните на листинг 9.1, который является посвящением криптографу Лену Сэссэмэну (Sassaman), автором которого предположительно является Дэн Камински (Dan Kaminsky). (Сообщение приводится в три столбца для экономии места.)

Листинг 9.1. Посвящение в транзакции

```
---BEGIN TRIBUTE---
#./BitLen
::::::::::::::::::
:::::::.::.::.:.:::
:.: :.' ' ' ' ' : :
:.:’’ ,,xiW,"4x, ‘’
:  ,dWWWXXXXi,4WX,
‘ dWWWXXX7"     ‘X,
lWWWXX7    __   _ X
:WWWXX7 ,xXX7’ "^^X
lWWWX7, _.+,, _.+.,
:WWW7,. ‘^"-" ,^-’
 WW",X:        X,
 "7^^Xl.    _(_x7’
 l ( :X:       __ _
 ‘. " XX  ,xxWWWWX7
  )X- "" 4X" .___.
,W X     :Xi _,,_
WW X      4XiyXWWXd
"" ,,      4XWWWWXX
, R7X,       "^447^
R, "4RXk,      _, ,
TWk  "4RXXi,   X’,x
lTWk,  "4RRR7’ 4 XH
:lWWWk,  ^"     ‘4
::TTXWWi,_  Xll :..
=-=-=-=-=-=-=-=-=-=
```

```
 LEN "rabbi" SASSAMA
      1980-2011
Len was our friend.
A brilliant mind,
 a kind soul, and
 a devious schemer;
 husband to Meredith
 brother to Calvin,
 son to Jim and
 Dana Hartshorn,
 coauthor and
 cofounder and
 Shmoo and so much
 more. We dedicate
 this silly hack to
 Len, who would have
 found it absolutely
 hilarious.
 --Dan Kaminsky,
 Travis Goodspeed
```

```
P.S. My apologies,
BitCoin people. He
also would have
LOL'd at BitCoin's
new dependency upon[1]
   ASCII BERNANKE
:’::.:::::.:::.::.:
: :.: ‘ ‘ ‘ ‘ : :’:
:.:     _.__    ‘.:
:   _,^"   "^x,   :
‘  x7’       ‘4,
 XX7           4XX
 XX             XX
 Xl ,xxx,   ,xxx,XX
( ‘ _,+o, | ,o+,"
 4   "-^’ X "^-’" 7
 l,     ( ))     ,X
 :Xx,_ ,xXXXxx,_,XX
  4XXiX’-___-’XXXX’
   4XXi,_   _iXX7’
   , ‘4XXXXXXXXX^ _,
   Xx,  ""^^^XX7,xX
W,"4WWx,_ _,XxWWX7’
Xwi, "4WW7""4WW7’,W
TXXWw, ^7 Xk 47 ,WH
:TXXXWw,_ "), ,wWT:
::TTXXWWW lXl WWT:
----END TRIBUTE----
```

Эта, безусловно, интересная и забавная проделка имела некоторые последствия для полных узлов Биткоин.

Сообщение в листинге 9.1 было записано в блокчейне с использованием одной транзакции с идентификатором

```
930a2114cdaa86e1fac46d15c74e81c09eee1d4150ff9d48e76cb0697d8e1d72
```

Его автор создал транзакцию с 78 выходами, по одному для каждой 20-символьной строки. Каждая строка завершается пробелом, поэтому видимы только 19 символов.

Например, вот как выглядит сценарий открытого ключа в последнем выходе:

```
OP_DUP OP_HASH160 2d2d2d2d454e4420545249425554452d2d2d2d20
↳OP_EQUALVERIFY OP_CHECKSIG
```

[1] Перевод: «Лен был нашим другом. Хитрый интриган, блестящий ум и добрая душа. Муж Мередит, брат Калвина, сын Джима и Даны Хартшорн. Соавтор, соучредитель и “Шму”. Мы посвящаем эту глупую проделку Лену, который наверняка нашел бы ее очень забавной. — Дэн Камински, Трэвис Гудспид. P. S. Мои извинения биткоин-сообществу, но Лен всегда смеялся насчет зависимости Биткоин от этого человека».

Наибольший интерес представляет PKH. Это не настоящий PKH. Возможно, вы заметите закономерность, сравнив его со строкой «`----END TRIBUTE----` »:

```
2d 2d 2d 2d 45 4e 44 20 54 52 49 42 55 54 45 2d 2d 2d 2d 20
-  -  -  -  E  N  D     T  R  I  B  U  T  E  -  -  -  -
```

Этот «хеш открытого ключа» кодирует одну 20-символьную строку в сообщении. Для представления символов в нем используются *ASCII-коды*. Например, символ «дефис» (`-`) представлен байтом `2d`. Символы A–Z представлены байтами `41`–`5a`, а пробел — байтом `20`.

ОБОЗРЕВАТЕЛЬ БЛОКЧЕЙНА

Вы можете сами исследовать эту транзакцию, использовав обозреватель блокчейна, который можно получить на веб-ресурсе 17 (приложение B).

Давайте рассмотрим PKH последних 10 строк сообщения вместе с декодированным текстом ASCII:

```
202034585869582720d5f5f5f2d60585858582720   4XXiX’-___-’XXXX’
202020345858692c5f2020205f69585837272020    4XXi,_ _iXX7’
20202c2060345858585858585858585e205f2c20   , ‘4XXXXXXXXX^ _,
202058782c202022225e5e5e5858372c78582020   Xx,  ""^^^XX7,xX
572c22345757782c5f205f2c5878575758372720 W,"4WWx,_ _,XxWWX7’’
5877692c2022345757372222234575737272c5720 Xwi, "4WW7""4WW7’,W
54585857772c205e3720586b203437202c574820 TXXWw, ^7 Xk 47 ,WH
3a5458585857772c5f2022292c202c7757543a20 :TXXXWw,_ "), ,wWT:
3a3a54545858575757206c586c205757543a2020 ::TTXXWWW lXl WWT:
2d2d2d2d454e44205452494254452d2d2d2d20 ----END TRIBUTE----
```

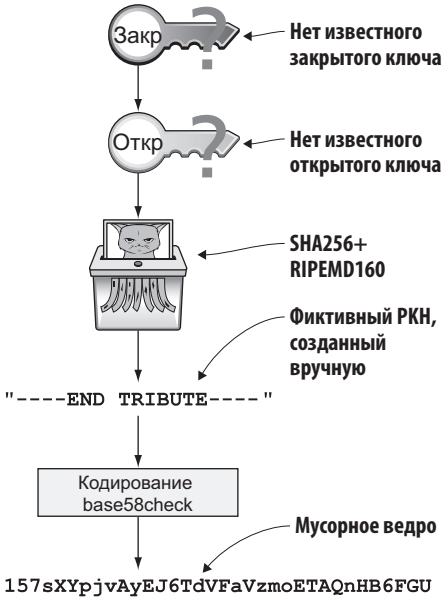
Раздутый набор UTXO

Поскольку эти PKH создаются вручную, они не имеют известных исходных образов. Это также означает, что с ними не связаны никакие известные пары открытых/закрытых ключей и никто не сможет потратить выходы этой транзакции. Они *не могут быть израсходованы*. Последний биткоин-адрес PKH — `157sXYpj...QnHB6FGU`. Любой, кто отправит деньги на этот адрес, выбросит их в мусорное ведро. Деньги будут потеряны навсегда. Это как сжечь долларовую банкноту.

Нерасходуемые выходы, подобные этим, неотличимы от обычных выходов, которые можно израсходовать. Вы не сможете доказать, что они нерасходуемые. Полные узлы вынуждены обрабатывать их как расходуемые, то есть они вынуждены навсегда сохранить эти нерасходуемые выходы в своих наборах неизрасходованных выходов транзакций (Unspent Transaction

Outputs, UTXO). Это накладывает ненужное бремя на узлы, которые обязаны хранить все эти выходы в памяти.

Разработчики Биткоин придумали частичное решение этой проблемы. Вместо отправки денег на недоказуемо нерасходуемые выходы пользователи могут создавать *доказуемо нерасходуемые* выходы. Если полный узел сможет определить, что выход является нерасходуемым, он не обязан добавлять этот выход в свой набор UTXO.



Это частичное решение основано на новом операторе `OP_RETURN`. При выполнении этот оператор сразу же завершается с ошибкой. Типичный сценарий открытого ключа с оператором `OP_RETURN` может выглядеть так:

```
OP_RETURN "I'm Grokking Bitcoin"
```

Если кто-то попытается потратить этот выход, сценарий завершится ошибкой, встретив `OP_RETURN`. Если сценарий открытого ключа содержит этот оператор, полный узел сможет определить, что выход является нерасходуемым, и игнорировать его, предотвратив раздувание набора UTXO. Типичный выход `OP_RETURN` платит 0 BTC, но он также может иметь значение больше 0, чтобы «сжечь» деньги.

Применение оператора `OP_RETURN` предполагает следование нескольким политикам:

- полный сценарий открытого ключа не должен иметь размер больше 83 байт;
- транзакция может иметь только один выход `OP_RETURN`.

Но это всего лишь политики. Полные узлы, придерживающие их, не будут пересылать транзакции, нарушающие эти политики. Но получив блок, содержащий транзакции, которые нарушают эти политики, они примут его и передадут дальше. В главах 10 и 11 я подробнее остановлюсь на политиках и *правилах консенсуса*, строгих правилах, применяемых к блокам.

Создание электронных меток в Биткоин

В главе 1 я уже упоминал о возможности с помощью блокчейна отслеживать право на владение. Предположим, что производитель автомобилей, назовем его Ampere, решил с помощью блокчейна Биткоин организовать цифровое отслеживание владения своими машинами. Сделать это можно, создав электронную метку.

Предположим, что в Ampere решили создать бирку для вновь произведенного автомобиля с номером шасси 123456. Для этого они отправляют Биткоин-транзакцию, изображенную на рис. 9.14.

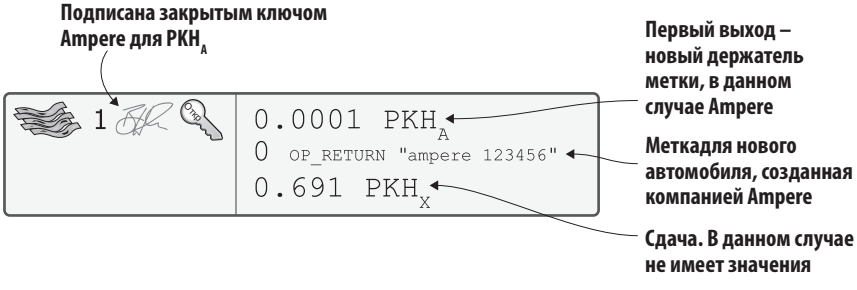


Рис. 9.14. В Ampere создали метку для вновь произведенного автомобиля. Они посылают бирку самим себе, потому что в данный момент именно они владеют автомобилем

Этот «протокол меток Ampere» определяет, что новая метка создается, когда:

- Ampere тратит средства с адреса PKH_A;
- транзакция содержит выход `OP_RETURN` с текстом `"ampere <номер шасси>"`;
- первый выход представляет первоначального владельца метки.

Компания Ampere имеет широко известную веб-страницу по адресу *https://www.ampere.example.com*, где опубликовала свой открытый ключ, соответствующий PKH_A. Она также опубликовала свой открытый ключ в рекламных объявлениях, в Facebook и Twitter. Все это сделано для того, чтобы люди могли убедиться, что PKH_A действительно принадлежит компании Ampere.

Предположим, что Ampere продала этот автомобиль дилеру. У дилера есть свой открытый ключ с хешем PKH_D. На рис. 9.15 показано, как Ampere передаст дилеру право владения в цифровом виде.

Рис. 9.15. Ampere продает автомобиль дилеру с хешем открытого ключа PKH_D

Согласно этому простому протоколу, владение автомобилем передается за счет расходования выхода предыдущего владельца. При этом применяются следующие правила:

- транзакция расходует выход предыдущего владельца;
- первый выход расходующей транзакции представляет нового владельца автомобиля.

Теперь дилер является новым владельцем, потому что в первом выходе расходующей транзакции указан его адрес PKH_D. Вот и все! Когда дилер продаст этот автомобиль клиенту, например Фатиме, он передаст право владения автомобилем на адрес Фатимы, PKH_F (рис. 9.16).

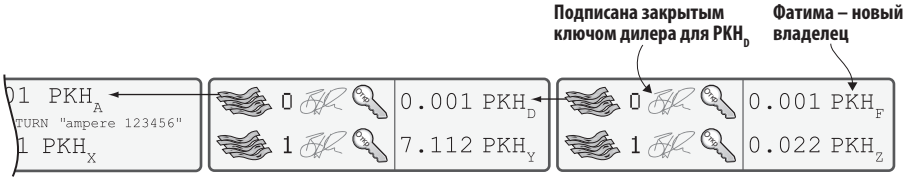


Рис. 9.16. Дилер передает право владения автомобилем на адрес Фатимы PKH_F

Запуск двигателя автомобиля с доказательством права владения

Теперь, когда Фатима является законным владельцем автомобиля, было бы здорово, если бы она могла завести его, доказав, что является владельцем. Такое тоже возможно. Автомобиль оборудован замком зажигания, который запускает двигатель, когда Фатима посылает подтверждение владения автомобилем (рис. 9.17).

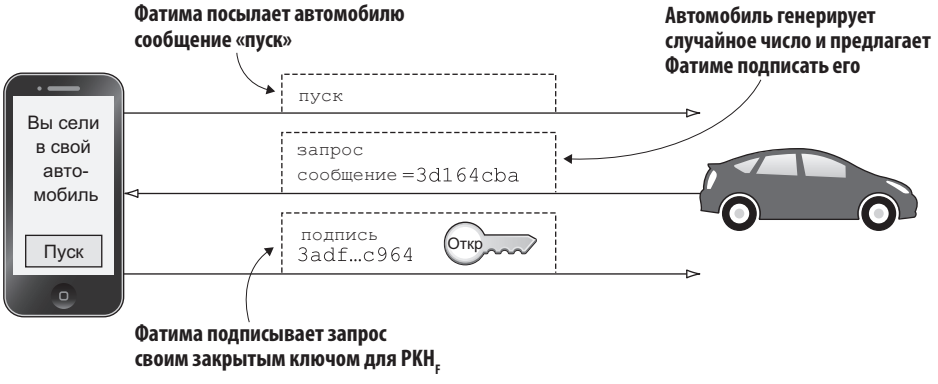


Рис. 9.17. Фатима заводит автомобиль, подписав запрос своим закрытым ключом

Сначала Фатима дает команду запустить двигатель. Автомобиль не заведется, пока не будет доказано, что у Фатимы есть закрытый ключ, принадлежащий РКН$_F$. Автомобиль генерирует большое случайное число и посылает его Фатиме. Она подписывает число своим закрытым ключом и отправляет подпись со своим открытым ключом автомобилю.

Открытый ключ нужен автомобилю, чтобы убедиться, что он соответствует хешу РКН$_F$, записанному в блокчейн. Автомобиль следит за тем, кому он принадлежит в данный момент, используя легкий кошелек, понимающий протокол бирок Ampere.

Когда автомобиль убедится, что подпись действительна и соответствует правильному закрытому ключу, он запустит двигатель.

Замена ожидающих транзакций

Когда вы отправляете Биткоин-транзакцию для покупки книги через Интернет, книжный магазин сначала дожидается подтверждения транзакции, и только потом отправляет книгу. Обычно транзакции подтверждаются в течение часа, но что, если это не так? Что, если ни один майнер не захочет включить вашу транзакцию в блок? Такое вполне может произойти, если комиссия за транзакцию недостаточно велика (рис. 9.18).

Как рассказывалось в разделе «Комиссионные отчисления за транзакции» в главе 7, комиссионные отчисления за транзакцию — это сумма входов минус сумма выходов. Плата за байт, которую получит майнер, вычисляется

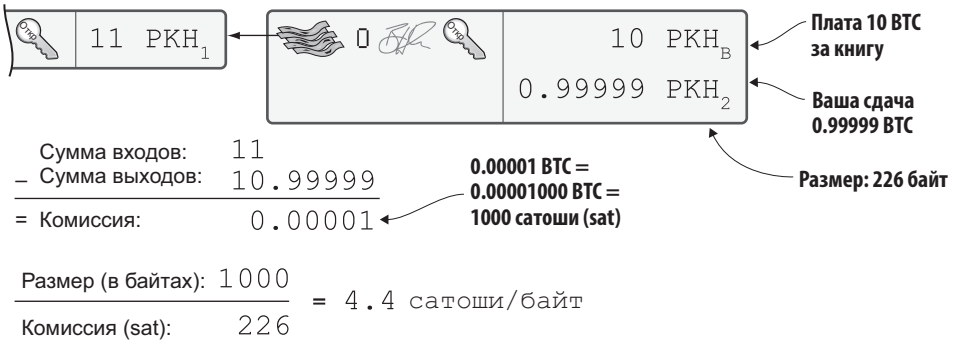


Рис. 9.18. Вы заплатили за книгу и установили размер комиссии равным 0.00001 BTC

делением комиссии на размер транзакции — в данном случае 1000 сатоши делятся на 226 байт и в результате получается примерно 4.4 сатоши за байт.

Если ни один майнер не захочет включить вашу транзакцию в блок за эту плату, она зависнет в ожидании подтверждения. Пока транзакция не будет подтверждена, вы не получите свою книгу. Вполне возможно, что вы захотите как-то повлиять на эту ситуацию, например создать новую похожую транзакцию, но с более высокой комиссией. Давайте попробуем (рис. 9.19).

Итак, вы создали и подписали новую транзакцию с комиссией в 20 раз больше. Как вам кажется, майнеры должны заинтересоваться такой транзакцией и быстро подтвердить ее.

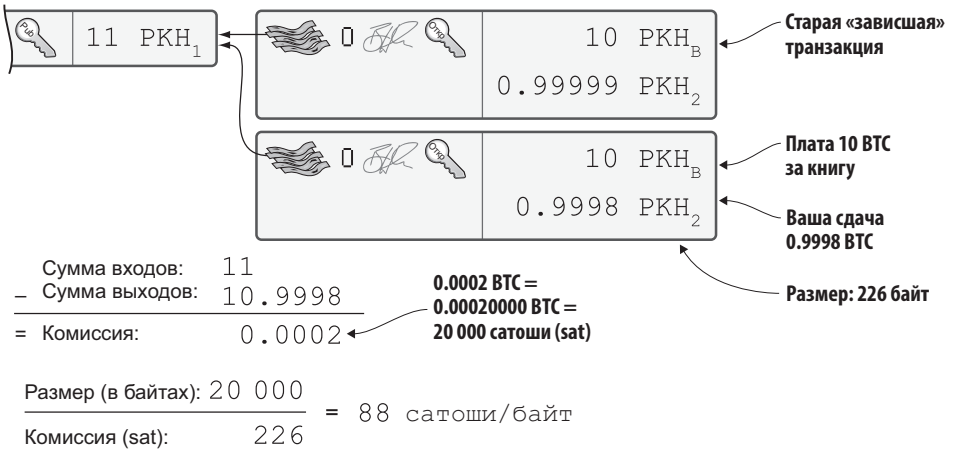


Рис. 9.19. Вы попытались заменить старую, «зависшую» транзакцию новой, с более высокой комиссией

Проблема в том, что большинством узлов новая транзакция, скорее всего, будет воспринята как попытка повторного расходования и отброшена. Они посчитают верной первую транзакцию и проигнорируют любые другие транзакции, расходующие тот же выход. Обрабатывать или не обрабатывать вторую транзакцию, узлы решают сами, но большинство из них, следуя общепринятым правилам, просто отбросят ее. Именно так работает Bitcoin Core, а это наиболее широко используемое программное обеспечение в системе Биткоин. Это правило называется *правилом первой встреченной транзакции*.

ПОДСКАЗКА ДЛЯ УПРАЖНЕНИЙ

Запомните эту информацию, она пригодится вам в упражнении 11.

Вы можете обойти это правило, отправив вторую транзакцию напрямую одному или нескольким майнерам. Майнеры движимы другими стимулами, нежели полные узлы. Полные узлы, осуществляющие майнинг, хотят получать вознаграждение — субсидию плюс комиссионные — за найденное доказательство работы, в то время как полные узлы, не занимающиеся майнингом, стремятся снизить потребление памяти и вычислительных ресурсов. Получив вторую транзакцию с высокой комиссией, майнер может решить включить ее в блок, несмотря на то что первой встреченной была транзакция с низкой комиссией. Замена транзакций таким способом нецелесообразна, поскольку вы не знаете IP-адреса майнеров, если они не опубликованы каким-либо образом. Кроме того, вам придется сообщить майнеру свой IP-адрес, а майнер может стать объектом интереса для различных компаний или надзорных организаций, желающих купить информацию о вас.

Согласие на замену за плату

В 2016 году было принято правило замены транзакций. Обычно его называют *согласием на замену за плату* (opt-in replace-by-fee, или opt-in RBF), см. рис. 9.20. Оно основано на использовании порядковых номеров входов транзакций.

Предположим еще раз, что вы решили купить книгу в онлайн-магазине. Создавая транзакцию, вы должны убедиться, что один из входов (в этом примере он единственный) имеет порядковый номер меньше, чем `fffffffe`. Этот номер сигнализирует узлам, что транзакция является заменяемой.

BIP125

В этом предложении по улучшению описывается, как транзакции могут объявлять себя заменяемыми.

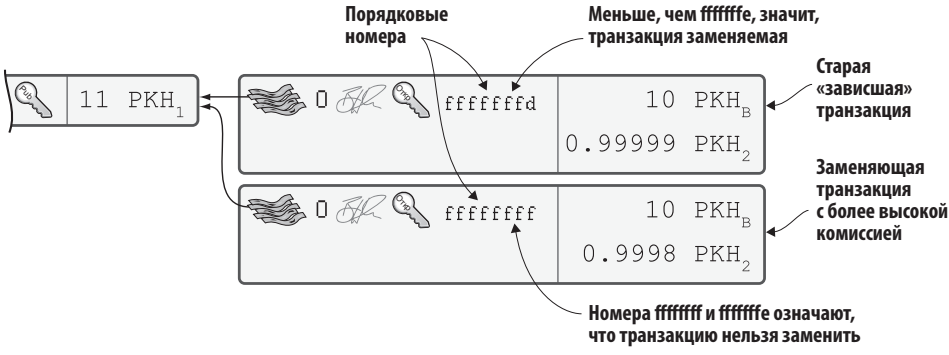


Рис. 9.20. Использование согласия на замену за плату для замены транзакции до ее подтверждения

Получив эту транзакцию, узел будет рассматривать ее как обычную транзакцию, но запомнит возможность замены.

Когда позднее вы заметите, что транзакция не подтверждается из-за слишком низкой комиссии, вы сможете создать новую заменяющую транзакцию с более высокой комиссией и отправить ее. Получив заменяющую транзакцию, узлы, если они реализуют правило opt-in RBF, любезно заменят старую транзакцию новой и разошлют новую транзакцию своим соседям. Старая транзакция будет удалена. В конечном итоге заменяющая транзакция достигнет всех узлов, включая майнеров, и, как мы надеемся, будет подтверждена в течение разумного периода времени.

В этом примере мы установили порядковый номер входа в заменяющей транзакции равным `ffffffff`. Это означает, что сама заменяющая транзакция не подлежит замене. Если вы хотите иметь возможность заменить эту заменяющую транзакцию, вы должны установить ее порядковый номер равным `fffffffd` или меньше, как вы делали это с замененной транзакцией.

Возможно, вам интересно узнать, откуда взялись эти порядковые номера. Первоначально порядковые номера предназначались для учета другого вида замены транзакций. Но от этой возможности отказались на ранней стадии развития Биткоин, однако порядковые номера во входах транзакций остались. С тех пор эти порядковые номера стали использоваться для временной блокировки по абсолютному значению и для замены за плату, как описано в этой главе. Если вы чувствуете, что начинаете путаться, не волнуйтесь; я кратко изложу разные варианты использования порядковых номеров в разделе «Повторение» в конце этой главы.

Дети платят за родителей

Есть еще один способ увеличить комиссионные за транзакцию. Предположим, у вас сложилась ситуация, изображенная ранее на рис. 9.18, и вы заметили, что эта транзакция зависла (рис. 9.21).

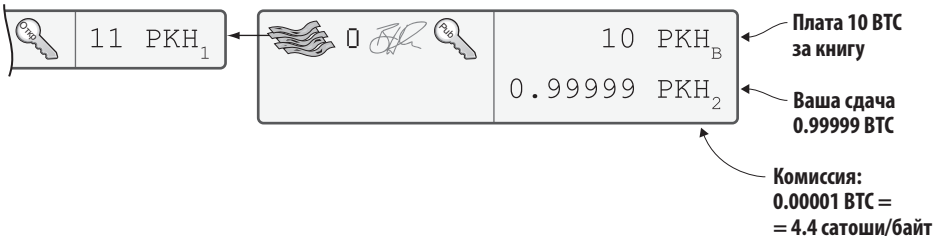


Рис. 9.21. Сумма комиссии оказалась недостаточно высокой. Транзакция зависла, потому что майнеры не заинтересовались ее включением в блок

Вы можете создать другую транзакцию, которая потратит вашу сдачу и заплатит более высокую комиссию, чтобы компенсировать низкую комиссию в исходной транзакции (рис. 9.22).

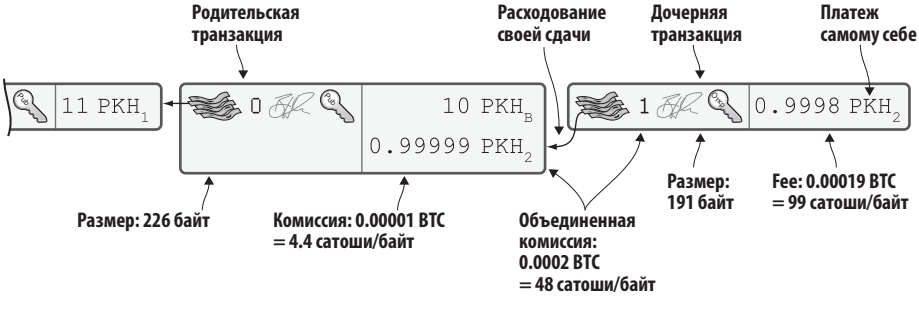


Рис. 9.22. Расходование сдачи и выплата дополнительной комиссии за «родительскую» транзакцию

Предположим, что майнер увидел эти две транзакции. Чтобы получить комиссию из дочерней транзакции, он должен включать в блок обе транзакции — родительскую и дочернюю. Если он включит только дочернюю транзакцию, блок станет недействительным, потому что дочерняя транзакция тратит деньги, отсутствующие в блокчейне.

Вы и книжный магазин — оба — можете выполнить этот трюк. Комиссию можете увеличить не только вы, но и книжный магазин, потратив

10 BTC и заплатив самому себе 9,9998 BTC, чтобы добавить 0,0002 BTC к комиссии.

Разные типы подписей

Подписывая обычную транзакцию, вы подписываете всю транзакцию, кроме сценария подписи (рис. 9.23).

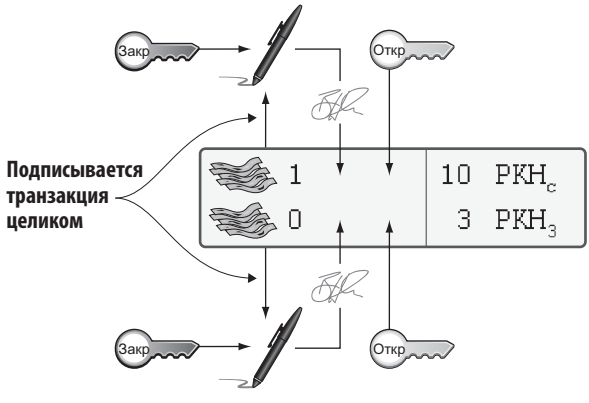


Рис. 9.23. Обычно подписывается транзакция целиком, включая все ее входы и выходы

Транзакция на рис. 9.23 содержит два входа, и каждый из них подписывает всю транзакцию целиком. Подпись *удостоверяет* все входы и все выходы. Если какой-либо из входов или выходов изменится, подпись станет недействительной.

Есть возможность ограничить область действия подписи, если использовать параметр в подписи, который называют типом `SIGHASH`. Удостоверить выходы можно тремя способами (`ALL`, `SINGLE` и `NONE`), а входы — двумя (наличие или отсутствие `ANYONECANPAY`). Допускаются любые комбинации входных и выходных типов `SIGHASH`, то есть всего возможно шесть различных комбинаций, как показано на рис. 9.24.

Для выходов можно удостоверить:

- *Все выходы* (`ALL`) — никто не сможет изменить какие-либо выходы.
- *Единственный выход с тем же индексом, что и вход* (`SINGLE`), — удостоверяется только один конкретный выход. Остальные выходы могут изменяться.
- *Ни одного выхода* (`NONE`) — вам все равно, куда уйдут деньги. Любой сможет добавить свои выходы и не сделать подпись недействительной.

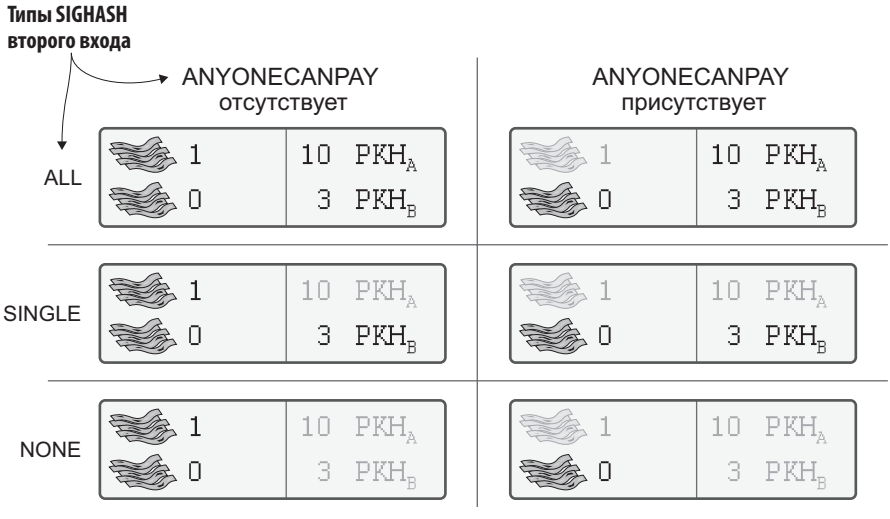


Рис. 9.24. Подпись может удостоверять разные части транзакции, в зависимости от типов SIGHASH. Подпись не охватывает элементы, выделенные серым цветом

Для входов можно удостоверить:

- *Все входы* (`ANYONECANPAY` отсутствует) — никто не сможет изменить какие-либо входы, не сделав подпись недействительной.
- *Только текущий вход* (`ANYONECANPAY` присутствует) — другие входы могут изменяться, удаляться или добавляться. Вам не интересно, кто заплатит. Заплатить сможет любой.

В подавляющем большинстве случаев используются подписи с типом `ALL` в сочетании с отсутствием `ANYONECANPAY`, удостоверяющие всю транзакцию целиком. Именно этот способ мы использовали в предыдущих главах книги. Другие типы редки и в основном используются для специализированных цифровых контрактов.

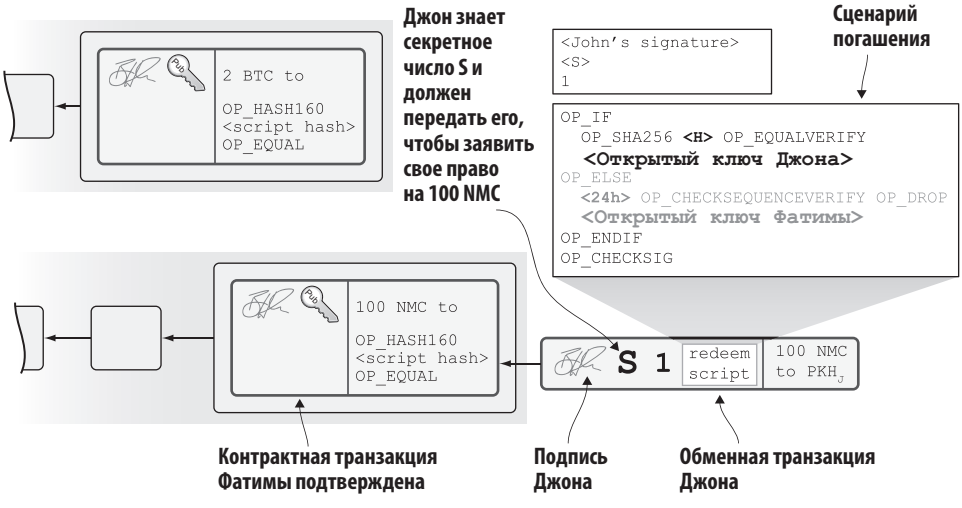
Повторение

В этой главе рассказывалось о том, что можно делать с транзакциями.

Транзакции и их выходы можно заблокировать на время разными способами, чтобы предотвратить расходование средств до определенной даты или до истечения определенного промежутка времени, как показано в следующей таблице.

Действие	Результат
Установка временной блокировки на транзакцию	Транзакция будет оставаться недействительной до определенного момента времени или высоты блока
Установка относительной временной блокировки на вход с использованием порядкового номера	Транзакция будет оставаться недействительной до истечения определенного промежутка времени или создания определенного количества блоков
Применение оператора `OP_CHECKLOCKTIMEVERIFY` в сценарии открытого ключа	Выход нельзя будет потратить до определенного момента времени или высоты блока
Применение оператора `OP_CHECKSEQUENCEVERIFY` в сценарии открытого ключа	Выход нельзя будет потратить до истечения определенного промежутка времени или создания определенного количества блоков

Во всех этих случаях продолжительность блокировки можно выразить высотой блока или значением времени. В основном временные блокировки полезны в цифровых контрактах, таких как атомарный обмен. Атомарный обмен позволяет людям, не доверяющим друг другу, обменивать монеты без привлечения третьей доверенной стороны.



Основная идея в том, что Джон должен раскрыть секретное число `S`, чтобы получить свои монеты. После этого Фатима сможет использовать `S`, чтобы получить свои монеты.

В выходах с оператором `OP_RETURN` можно хранить произвольные данные, не увеличивая нагрузку на наборы UTXO узлов. Эту особенность можно использовать для создания бирок. Например, с помощью блокчейна Биткоин можно отслеживать и проверять право собственности на автомобиль.

Иногда транзакция может зависнуть в состоянии ожидания, если майнеры не захотят включить ее в свои блоки. Обычно это происходит из-за недостаточно высокой комиссии. Чтобы получить возможность повлиять на эту ситуацию, можно пометить транзакцию как заменяемую, установив хотя бы в одном входе порядковый номер меньше `fffffffe`. Если такая транзакция зависнет, вы сможете увеличить комиссию, послав заменяющую ее транзакцию с более высокой комиссией.

Порядковые номера во входах можно использовать для разных целей. В этой главе мы обсудили несколько возможных вариантов, но их трудно запомнить. Чтобы поправить ситуацию, в табл. 9.1 описывается, какое влияние оказывают разные значения порядковых номеров.

Таблица 9.1. Порядковые номера управляют включением/выключением разных особенностей

Значение порядкового номера	Временная блокировка, любой вход	Замена за плату (BIP125), любой вход	Относительная временная блокировка входа (BIP68)*
`00000000-7fffffff`	✓	✓	✓
`80000000-fffffffd`	✓	✓	×
`fffffffe`	✓	×	×
`ffffffff`	×	×	×

✓ — включено; × — выключено.

* Транзакция должна иметь версию не ниже 2.

Упражнения

Для разминки

9.1. Что необходимо указать во входах транзакции, чтобы заблокировать ее до определенной отметки времени?

9.2. Допустим, что на транзакцию наложена временная блокировка (абсолютная), действие которой истекает в 00:00:00 25 декабря 2019 года. Как майнер проверит, что ее можно включить в блок?

9.3. Где определяется относительная временная блокировка входа?

9.4. Предположим, что Адам и Ева хотят обменяться монетами друг с другом, используя атомарный обмен. Сколько транзакций будет создано в каждой цепочке блоков по завершении?

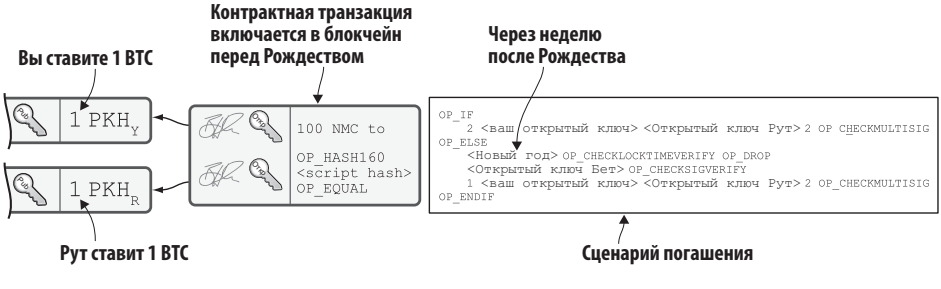
9.5. Чем плохо для набора UTXO хранение произвольных данных, таких как «HELLO WORLD», в виде фиктивных PKH в выходах, вместо сохранения их в выходах с оператором `OP_RETURN`?

9.6. В каких случаях может появиться желание заменить отправленную, но еще не подтвержденную транзакцию?

Придется пораскинуть мозгами

9.7. Объясните разницу между абсолютной и относительной временными блокировками.

9.8. (Это трудное упражнение; можете пропустить его.) Представьте, вы решили поспорить на 1 BTC, что в канун Рождества в Лондоне будет идти снег. Рут поддержала спор, заявив, что этого не случится. Для разрешения любых возможных конфликтов вы назначаете арбитром Бет, которой оба доверяете. Вы и Рут создаете и посылаете транзакции, которые переводят по 1 BTC на выход с 2 BTC со следующим сценарием погашения. (Сценарий погашения *можно* уменьшить, но чтобы его было проще читать, я использовал расширенную версию.) Объясните на концептуальном уровне, как работает этот сценарий погашения.



9.9. Если выход p2sh переводит средства на хеш сценария погашения, состоящий только из оператора `OP_RETURN` с 32 случайными байтами, смогут ли полные узлы определить, что выход является нерасходуемым?

```
OP_RETURN 53a1e411...b4e6d949
```

9.10. Объясните, как работает правило первой встреченной транзакции. Обязаны ли узлы следовать ему?

9.11. Согласие на замену за плату (opt-in RBF) — это метод замены транзакции. Есть ли принципиальная разница в безопасности между транзакциями с включенным и выключенным согласием на замену? Объясните свои рассуждения.

Итоги

- Транзакции можно блокировать по времени или высоте блока, в зависимости от конкретных потребностей. Блокировки могут быть абсолютными и относительными.
- Выход транзакции может потребовать заблокировать на время расходующую транзакцию. Эту возможность можно использовать во многих цифровых контрактах.
- Атомарный обмен — это способ обмена криптовалютами между двумя сторонами, не доверяющими друг другу.
- Произвольные данные — например, бирку, подтверждающую владение автомобилем, — можно хранить в выходах `OP_RETURN`, чтобы предотвратить раздувание набора UTXO.
- Транзакцию можно объявить заменяемой. Это позволит заменить транзакцию, если она не будет подтверждена в течение разумного периода времени.
- Подписи могут удостоверять разные части транзакций. Всего возможно шесть комбинаций типов `SIGHASH`. Это может пригодиться в некоторых цифровых контрактах.

10 Segwit

Эта глава охватывает следующие темы:

- ✓ проблемы Биткоин;
- ✓ удаление подписей из транзакций.

Система Биткоин далека от идеала. В ней есть несколько недостатков, которые еще предстоит устранить. В первом разделе этой главы мы обсудим некоторые из этих недостатков. К числу наиболее важных относятся *пластичность транзакций* (transaction malleability) и неэффективность проверки подписи. Мы уже упоминали проблему пластичности транзакций в разделе «Временная блокировка транзакций» главы 9, суть которой состоит в том, что кто-то со стороны может изменить транзакцию на этапе распространения, не нарушая ее действительность, что приведет к изменению ее идентификатора.

Решение этих проблем было представлено в 2015 году на конференции по масштабированию Биткоин. Это решение известно как Segregated witness, segwit, или *отделенный свидетель*, — довольно странное название для решения, суть которого заключается в выделении подписей из транзакций. Я опишу это решение подробно: оно предполагает изменение практически всех частей Биткоин, включая адреса, формат транзакций, формат блоков, локальное хранилище и сетевой протокол.

Поскольку решение segwit предполагает весьма масштабные изменения в Биткоин, внедрить его оказалось очень непросто, не нарушив работу сети. Оно было тщательно спроектировано так, чтобы старое программное обеспе-

чение продолжало работать и принимать транзакции и блоки с поддержкой segwit, хотя и без некоторых проверок.

Проблемы, решаемые с помощью segwit

В этом разделе мы обсудим проблемы, которые решит segwit.

Пластичность транзакций

Чтобы объяснить суть проблемы, вернемся к примеру из главы 9, в котором вы передавали своей дочери транзакцию с временной блокировкой. Перед тем как истечет время блокировки, вы должны отменить эту и создать новую транзакцию с временной блокировкой, как показано на рис. 10.1.

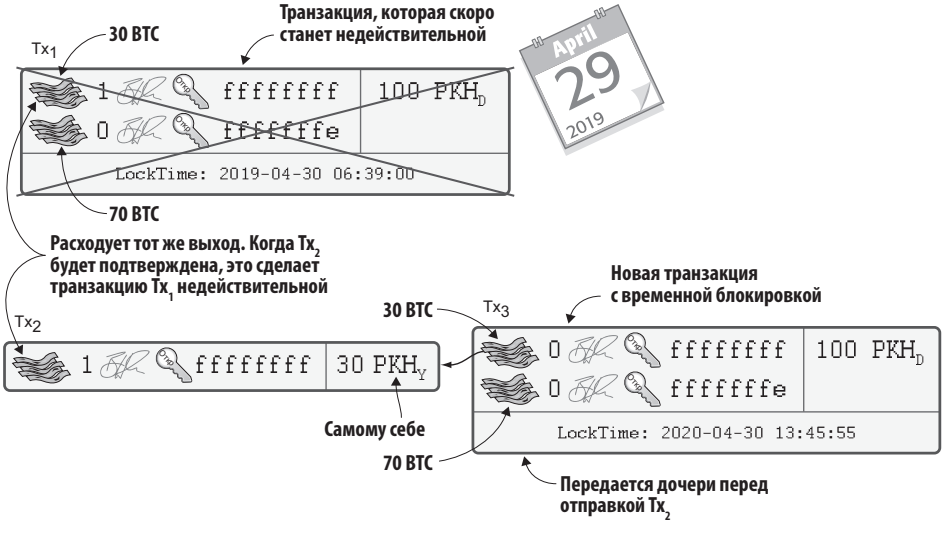


Рис. 10.1. Вы расходуете один из выходов, который должна потратить транзакция с временной блокировкой, и создаете новую транзакцию с временной блокировкой, посылая ее своей дочери

Новую транзакцию Tx_3 с временной блокировкой, которая аннулирует транзакцию Tx_1, важно передать дочери перед отправкой Tx_2. Если все сделать наоборот, вы можете попасть под автобус между аннулированием прежней и передачей новой транзакции, и тогда ваша дочь не получит деньги.

Предположим, вы все сделали правильно и сначала передали Tx_3 своей дочери, а затем отправили Tx_2. Транзакция Tx_3 расходует выход Tx_2, а значит, Tx_3 содержит идентификатор транзакции Tx_2 в одном из своих входов. Давайте посмотрим, что может произойти после отправки Tx_2 (рис. 10.2).

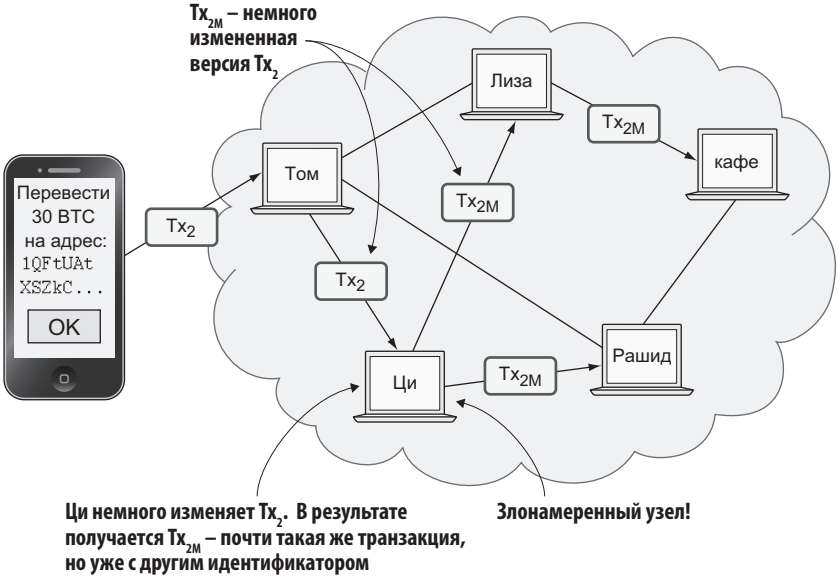


Рис. 10.2. Ци изменяет вашу транзакцию в процессе ее распространения по сети

Ци хочет все испортить. Получив вашу транзакцию Tx_2, она определенным образом изменяет ее и получает Tx_{2M}, которая остается действительной и имеет тот же эффект, что и исходная транзакция Tx_2. (Вскоре вы увидите несколько способов, как это сделать.) Но теперь по сети распространяются две разные транзакции, которые расходуют один и тот же выход и переводят одну и ту же сумму одному и тому же получателю, но имеют *разные идентификаторы*.

ПЛАСТИЧНОСТЬ

Слово *пластичность* означает возможность изменения формы. Например, металл — пластичный, и форму металлической детали можно изменить с помощью молотка. Этот термин используется в криптографии для обозначения изменения подписи без влияния на ее действительность или изменения зашифрованного сообщения без полного его искажения.

Поскольку Tx_2 и Tx_{2M} расходуют один и тот же выход, они вступают в конфликт друг с другом, поэтому подтверждена будет только одна из них. Предположим, что Tx_{2M} выиграла гонку и была включена в следующий блок. Что произойдет с наследством вашей дочери? Смотрите рис. 10.3.

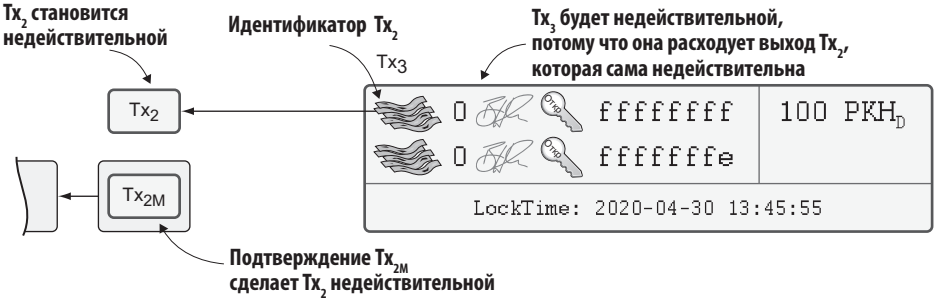


Рис. 10.3. Наследство будет потеряно, потому что транзакция, которая находится у вашей дочери, окажется недействительной из-за пластичности

Транзакция с *измененной формой*, Tx_{2M}, сохранится в блокчейне. Это сделает транзакцию Tx_2 недействительной, потому что она расходует тот же выход, что и Tx_{2M}. Первый вход транзакции с временной блокировкой, Tx_3, ссылается на Tx_2, используя ее идентификатор, поэтому после 30 апреля 2020 года ваша дочь не сможет претендовать на наследство: это будет расценено как попытка потратить выход недействительной транзакции.

Как Ци может изменить идентификатор транзакции?

Ци имеет на выбор несколько способов изменить транзакцию, не повлияв на ее действительность. Все они так или иначе связаны с изменением сценария подписи. На рис. 10.4 показаны три из них.

Первый способ основан на изменении формата контейнера подписи — способа *кодирования* подписи в сценарии подписи. Закодировать подпись можно несколькими разными способами, которые являются действительными. Эта проблема была исправлена после реализации BIP66, который требует, чтобы все подписи были закодированы определенным образом. Исправление было активировано в блоке 363724.

BIP66

Предложение по улучшению BIP66 исправило первый класс проблем, вызванных пластичностью транзакций.

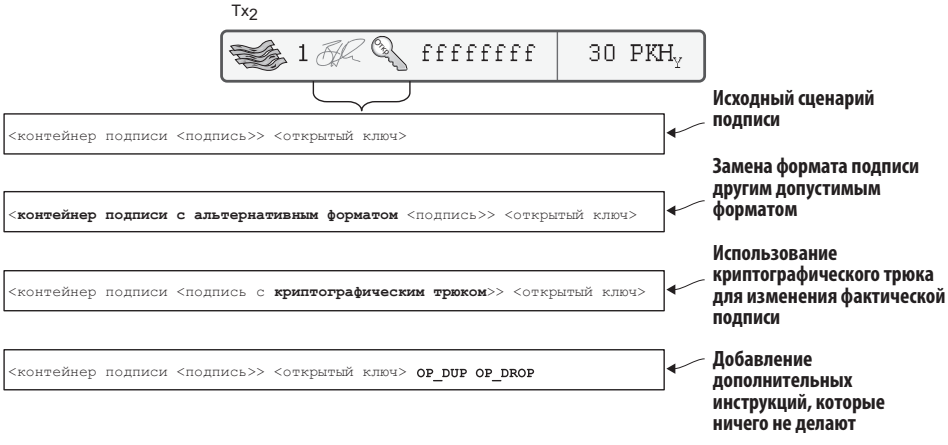


Рис. 10.4. Три способа изменить форму транзакции

Второй способ основан на использовании криптографических приемов. Я не буду вдаваться в подробности, но, независимо от формата контейнера, подпись можно изменить несколькими способами так, что она не утратит свою действительность. Пока известен только один такой трюк, но нельзя исключать, что есть и другие.

Последний способ заключается в изменении самого сценария. Сделать это можно по-разному. Один из вариантов показан на рис. 10.4: сначала на вершину стека помещается копия верхнего элемента (`OP_DUP`), а затем она удаляется (`OP_DROP`); по сути, это изменение ничего не делает, и сам сценарий будет работать нормально.

Второй и третий способы изменения формы транзакции несколько ограничены *правилами пересылки*, согласно которым узлы требуют, чтобы подписи соответствовали определенным правилам и сценарии подписи не содержали операторов, кроме тех, что помещают данные на стек. В противном случае узел не передаст такую транзакцию дальше. Но ничто не мешает майнеру попытаться включить такую транзакцию в блок. Правила пересылки усложняют распространение измененных транзакций, но не предотвращают возможности их включения в блоки.

Неэффективная проверка подписи

В процессе подписания транзакции алгоритм подписи определенным образом хеширует транзакцию.

Как говорилось в разделе «Подписывание транзакции» в главе 5, перед подписанием удаляются все сценарии подписи. Но если сделать *только* это, все подписи в транзакции будут использовать один и тот же хеш. Если транзакция использует два разных выхода и переводит средства на один и тот же адрес, подпись из одного входа можно будет повторно использовать во втором. Этой особенностью может воспользоваться злоумышленник.

Чтобы избежать этой проблемы, Биткоин заставляет каждую подпись удостоверять немного отличающуюся версию транзакции, которая получается копированием сценария открытого ключа из расходуемого выхода в сценарий подписи подписываемого входа.

ПОЧЕМУ БЫ НЕ ИСПОЛЬЗОВАТЬ ФИКТИВНЫЙ БАЙТ?

Копирование сценария открытого ключа в сценарий подписи выглядит слишком сложно. Проще добавить в сценарий подписи один фиктивный байт, чтобы исключить возможность повторного использования подписи. В действительности никто не знает, почему для этого используется именно сценарий открытого ключа.

Давайте рассмотрим поближе происходящее. Предположим, вы хотите подписать транзакцию с двумя входами. Первый подписан, как показано на рис. 10.5.

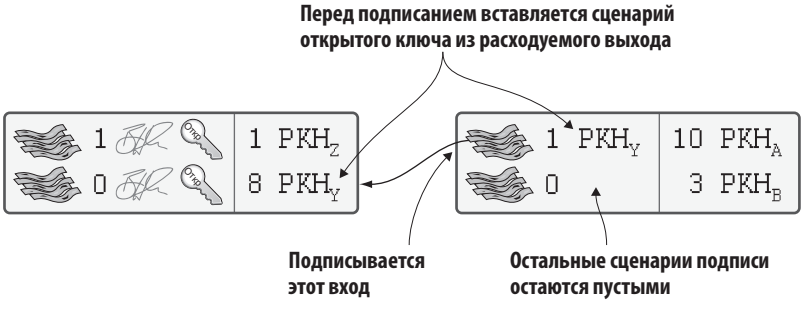


Рис. 10.5. Подписание первого входа. Сценарий открытого ключа копируется в сценарий подписи

Сценарии подписей во всех входах пустые, но вы копируете сценарий открытого ключа расходуемого выхода в сценарий подписи расходующего

входа. Затем подписываете первый вход и переходите к подписанию второго входа (рис. 10.6).

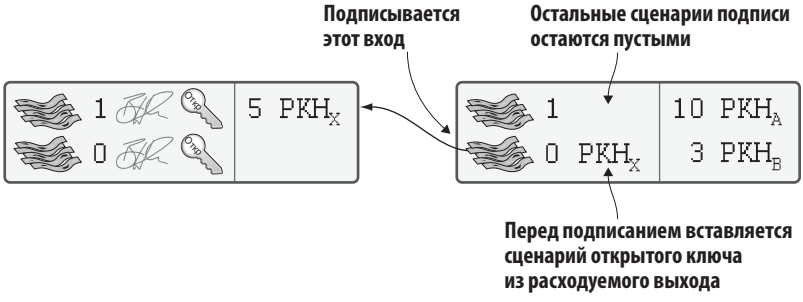


Рис. 10.6. Подписание второго входа

Здесь все сценарии подписей, кроме второго, пустые. Второй сценарий подписи заполняется сценарием открытого ключа из расходуемого выхода. Затем создается подпись.

Выполнение этой процедуры для каждого входа гарантирует, что подписи во входах не будут повторяться, если подписывать их одним и тем же закрытым ключом. Но это также создает проблему: проверка подписи становится неэффективной.

Предположим, вы решили проверить подписи в созданной выше транзакции. Для каждого входа необходимо выполнить практически ту же процедуру, что и при подписании транзакции: очистить все сценарии подписей в транзакции, затем вставить сценарий открытого ключа в сценарий подписи проверяемого входа и проверить его подпись.

Это усложнение может показаться незначительным, но с ростом числа входов увеличивается объем хешируемых данных для проверки каждой подписи. Если удвоить количество входов, тогда примерно:

- удвоится число проверяемых подписей;
- удвоится размер транзакции.

Если предположить, что для проверки транзакции с двумя входами на рис. 10.7 требуется 1 мс, тогда для проверки транзакции с четырьмя

ПОЧЕМУ 1 МИЛЛИСЕКУНДА?

Время 1 мс — это лишь пример. Фактическое время проверки транзакции меняется в зависимости от узла.

входами понадобится уже 4 мс. Удвойте количество входов, и вы получите 16 мс. Проверка транзакции с 1024 входами займет более 4 минут!

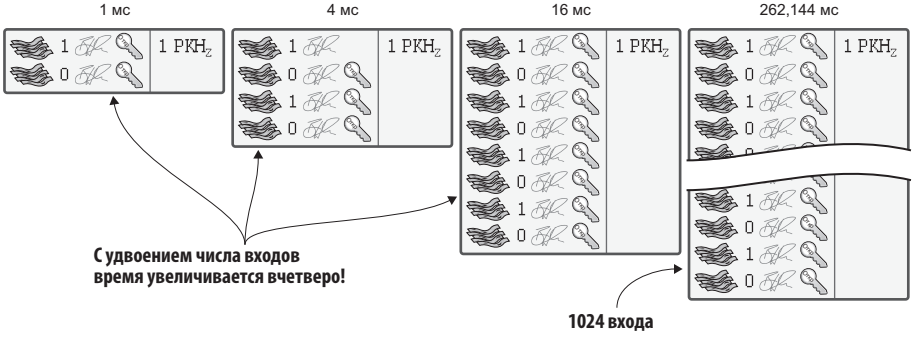


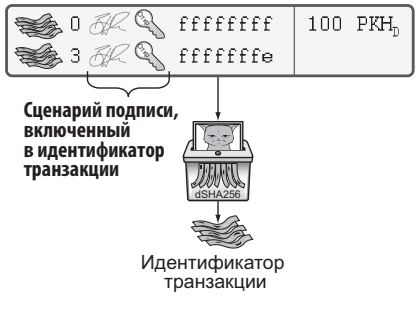
Рис. 10.7. Общее время на хеширование в ходе проверки подписей. С удвоением числа входов время увеличивается примерно вчетверо

Этой уязвимостью можно воспользоваться и создать большую транзакцию с большим количеством входных данных. Все узлы, проверяющие транзакцию, будут заняты в течение нескольких минут, что не позволит им проверять другие транзакции и блоки в это время. Сеть Биткоин замедлится.

Было бы лучше, если бы время проверки транзакции росло в линейной, а не в квадратичной зависимости: время проверки транзакции удваивалось с удвоением числа входов. Тогда для проверки 1024 входов потребуется около 512 мс вместо 4 минут.

Потеря пропускной способности

Посылая транзакцию в легкий кошелек, полный узел отправляет полную транзакцию, включая все подписи. Но легкий кошелек не может проверить подписи, потому что не имеет расходуемых выходов.



Сценарии подписи составляют значительный процент от размера транзакции. Типичный сценарий подписи, использующий выход p2pkh, занимает 107 байт. Рассмотрим несколько транзакций с двумя выходами (табл. 10.1).

Было бы лучше, если бы полному узлу не требовалось отправлять сценарии подписей в легкий кошелек. Тогда удалось бы сэкономить более 50% трафика. Есть только одна проблема: эти данные необходимы для вычисления идентификаторов транзакций. В их отсутствие легкий кошелек не сможет проверить включение транзакции в блок, потому что не сможет проверить доказательство Меркла (рис. 10.8).

Таблица 10.1. Пространство, занимаемое сценарием подписи в разных транзакциях

Входов	Общий размер сценариев подписей (байты)	Размер транзакции (байты)	Процент, занимаемый сценариями подписей
1	107	224	47%
2	214	373	57%
3	321	521	61%
8	856	1255	68%

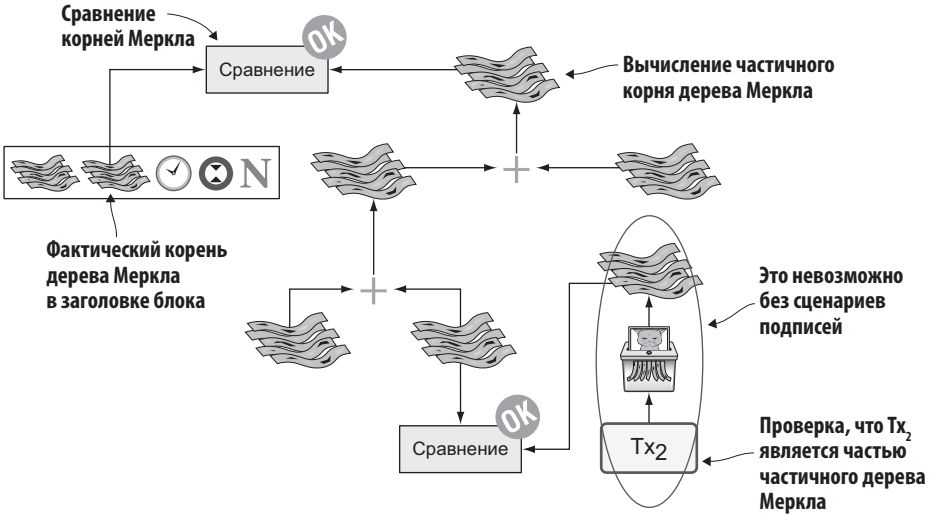


Рис. 10.8. Без сценариев подписей легкий кошелек не сможет проверить включение транзакции в блок

Мы определенно хотели бы найти решение этой проблемы.

Расширение языка сценариев — это сложно

Иногда возникает необходимость расширить язык сценариев новыми операциями. Например, в 2015 и 2016 годах в язык сценариев были введены операторы `OP_CHECKSEQUENCEVERIFY` (`OP_CSV`) и `OP_CHECKLOCKTIMEVERIFY` (`OP_CLTV`). Давайте посмотрим, как вводился оператор `OP_CLTV`.

Для начала разберемся, как выглядят операторы `OP_` в сценариях. Каждый из них занимает единственный байт. Оператор `OP_EQUAL`, например, представлен байтом `87` в шестнадцатеричном виде. Каждый узел знает, что встретив байт `87` в сценарии, он должен снять два верхних элемента со стека, сравнить их и положить результат обратно на стек. `OP_CHECKMULTISIG` тоже занимает один байт, `ae`. Все операторы представлены разными байтами.

Когда создавалась система Биткоин, в языке сценариев было определено несколько операторов NOP: `OP_NOP1` — `OP_NOP10`. Они представлены байтами `b0` — `b9`. Их назначение — представлять ничего не делающие операции. Название NOP происходит от No OPeration (нет операции), что фактически означает: «когда появляется эта инструкция, пропустить ее и двигаться дальше».

Эти операторы NOP могут использоваться для расширения языка сценариев, но только до определенной степени. Оператор `OP_CLTV` на самом деле — это бывший `OP_NOP2`, представленный байтом `b1`. Оператор `OP_CLTV` был введен в обращение в очередном выпуске Bitcoin Core, переопределившем `OP_NOP2`. Но сделать это было необходимо совместимым способом, чтобы не нарушить работоспособность со старыми, не обновленными узлами.

Вернемся к примеру из раздела «Временная блокировка выходов по абсолютному значению» в главе 9, где вы заранее перевели деньги своей дочери, которые она могла потратить 1 мая (рис. 10.9).

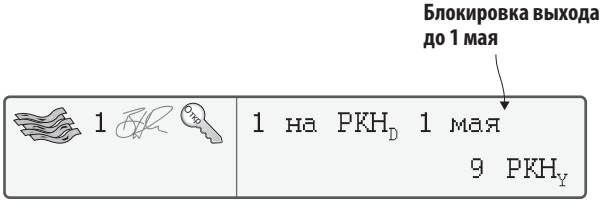


Рис. 10.9. Использование OP_CLTV для блокировки выхода до 1 мая

Вот как выглядит сценарий открытого ключа для этого выхода:

```
<1 may 2019 00:00:00> OP_CHECKLOCKTIMEVERIFY OP_DROP
OP_DUP OP_HASH160 <PKHD> OP_EQUALVERIFY OP_CHECKSIG
```

Вот как новый узел — который знает о новом значении байта `b1` — интерпретирует сценарий:

1. Положить на стек отметку времени `<1 may 2019 00:00:00>`.
2. Сравнить время блокировки в расходующей транзакции с отметкой времени на вершине стека, и если время в транзакции меньше времени на стеке, немедленно завершится с ошибкой.
3. Вытолкнуть значение времени со стека.
4. Продолжить обычную процедуру проверки подписи.

Старый узел интерпретирует тот же сценарий иначе:

```
<1 may 2019 00:00:00> OP_NOP2 OP_DROP
OP_DUP OP_HASH160 <PKHD> OP_EQUALVERIFY OP_CHECKSIG
```

1. Положить на стек отметку времени `<1 may 2019 00:00:00>`.
2. *Ничего не делать.*
3. Вытолкнуть значение со стека.
4. Продолжить обычную процедуру проверки подписи.

Старые узлы интерпретируют `OP_NOP2` как пустую операцию и движутся дальше. Они не знают новых правил, связанных с байтом `b1`.

Старый и новый узлы будут действовать одинаково, если проверка `OP_CLTV` на новом узле завершится успехом. Но если на новом узле `OP_CLTV` потерпит неудачу, то старый узел продолжит выполнение как ни в чем не бывало, потому что операция «ничего не делать» никогда не завершается ошибкой. Новые узлы будут чаще отказываться подтверждать транзакции, потому что следуют более строгим правилам. На старых узлах сценарий всегда будет завершаться успехом, если успехом завершается на новых узлах. Такое обновление системы, не требующее обновления всех узлов, называется *софт-форком* (soft fork). Подробнее о форках (ветвлениях), обновлениях системы и альтернативных валютах, появившихся из блокчейна Биткоина, мы поговорим в главе 11.

Возможно, вам интересно, для чего предназначена инструкция OP_DROP. Она просто снимает элемент с вершины стека и отбрасывает его. Инструкция OP_CLTV специально разрабатывалась так, чтобы соответствовать инструкции OP_NOP2 в случае успешного выполнения. Если бы OP_CLTV разрабатывалась без учета старых узлов, она, вероятно, удаляла бы верхний элемент со стека. Но из-за необходимости учитывать особенности работы старых узлов OP_CLTV этого не делает. Мы должны добавить дополнительный оператор OP_DROP после OP_CLTV, чтобы удалить элемент со значением времени из стека.

Это был пример того, как старые операторы могут перепрофилироваться для наложения дополнительных ограничений без нарушения работы всей сети.

До настоящего времени этот метод расширения языка сценариев применялся дважды:

Байт	Старый код	Новый код	Новое значение
b1	OP_NOP2	OP_CLTV	Убедиться, что расходующая транзакция имеет достаточно большое значение абсолютной временной блокировки
b2	OP_NOP3	OP_CSV	Убедиться, что расходующий вход имеет достаточно большое значение относительной временной блокировки

Для расширения языка сценариев доступно всего 10 операторов OP_NOP, и такие расширения должны точно имитировать поведение OP_NOP в случаях, когда завершаются успехом.

Рано или поздно свободные операторы OP_NOP закончатся, и понадобится другой механизм расширения языка сценариев, потому что новые операторы должны будут действовать иначе, чем OP_NOP, когда завершаются успехом.

Решения

На конференции 2015 года было предложено решение всех этих проблем: вынести сценарии подписей из транзакций.

Давайте посмотрим, как устроена обычная транзакция (рис. 10.10).

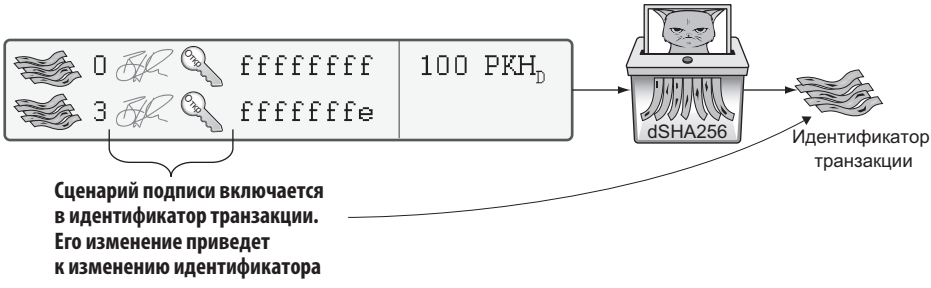


Рис. 10.10. Идентификатор транзакции вычисляется по всему ее содержимому, включая сценарии подписей

Изменив систему так, чтобы идентификатор транзакции вычислялся без учета сценариев подписей, можно было бы ликвидировать все известные причины пластичности транзакций. К сожалению, в этом случае старое программное обеспечение оказалось бы несовместимым с новым, потому что оно вычисляет идентификаторы транзакций традиционным способом.

BIP141

Новые правила, устанавливаемые решением Segwit, описаны в BIP141 «Segregated Witness (Consensus layer)».

Решение «отдельный свидетель» устраняет эту и все вышеупомянутые проблемы с сохранением прямой и обратной совместимости:

- прямая совместимость сохраняется, потому что блоки, созданные новым программным обеспечением, успешно обрабатываются старым программным обеспечением;
- обратная совместимость сохраняется, потому что блоки, созданные старым программным обеспечением, успешно обрабатываются новым программным обеспечением.

На языке криптографии под термином *witness* (свидетель) обычно подразумевается подпись — нечто, свидетельствующее о подлинности чего-либо. Свидетелем для транзакций в Биткоин является содержимое сценария подписи, потому что именно эти сценарии подтверждают аутентичность транзакций. Слово *segregated* (отделенный) означает отделение, поэтому мы отделяем содержимое сценариев подписей от транзакций, оставляя сценарии подписей пустыми, как показано на рис. 10.11.

Название segregated witness (отдельный свидетель) означает, что содержимое сценариев подписей вынесено из транзакций во внешнюю структуру, которая называется свидетелем (witness).

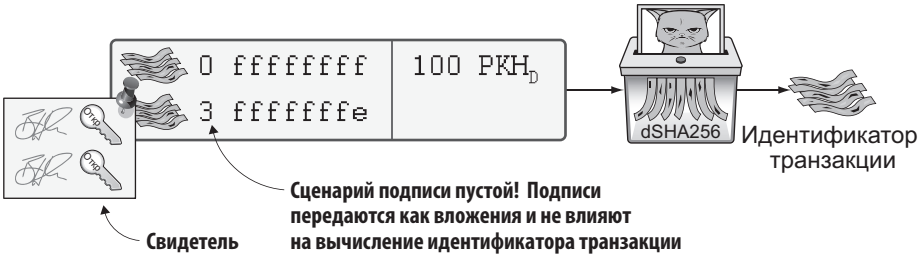


Рис. 10.11. Транзакция, соответствующая правилам segwit, не содержит подписей. Подписи сопровождают транзакцию отдельно. Идентификатор транзакции вычисляется без учета подписей

Давайте проследим за движением транзакций с отдельным свидетелем и посмотрим, как они влияют на различные части системы Биткоин. Но сначала получим немного биткоинов в segwit-кошелек.

Segwit-адреса

Предположим, ваш кошелек реализует решение segwit и вы решили продать ноутбук девушке Эми. Ваш кошелек должен создать адрес, чтобы вы могли передать его Эми. Пока ничего нового.

Но решение segwit определяет новый тип адреса, который кодируется с использованием алгоритма *Bech32* вместо base58check. Представим, что ваш кошелек создал следующий segwit-адрес:

BIP173

Это предложение по улучшению определяет схему кодирования Bech32 с контрольной суммой, а также порядок создания segwit-адресов и их кодирование с использованием Bech32.

```
bc1qeqzjk7vume5wmrdgz5xyehh54cchdjag6jdmkj
```

Этот формат адресов предлагает несколько усовершенствований, по сравнению с привычными адресами base58check:

- Все символы имеют один и тот же регистр, а значит:
  - QR-коды можно делать меньше;
  - адреса проще читаются.
- Контрольная сумма, используемая в Bech32, выявляет ошибки с длиной до четырех символов с вероятностью 100%. Если ошибка имеет большую

длину, вероятность пропустить ее составляет менее одной миллиардной. Это намного лучше 4-байтной контрольной суммы в base58check, которая не дает никаких гарантий.

Segwit-адрес состоит из двух частей. Первые два символа, `bc` (сокращение от bitcoin), — это *читаемая человеком часть*. Следующая далее цифра `1` — это разделитель между частью, читаемой человеком, и *фактическими данными*, которые Эми будет использовать для создания выхода транзакции:

- номер версии, в данном случае 0;
- *программа свидетеля*, в данном случае PKH `c8052b79...3176cba8`.

Что такое программа свидетеля, я расскажу чуть ниже, а пока просто считайте ее аналогом хеша открытого ключа (PKH). Версию и программу-свидетеля нельзя получить напрямую из адреса, потому что они закодированы с использованием алгоритма bech32. Вы передаете Эми адрес `bc1qeqzj...ag6jdmkj` в виде QR-кода. Она пользуется современным кошельком, который понимает адреса в этом формате, поэтому она сканирует ваш адрес и извлекает версию и программу свидетеля, как показано на рис. 10.12.

Делается это в несколько этапов:

❶ Часть с данными отделяется от части, предназначенной для чтения человеком.

❷ Часть с данными, символ за символом, преобразуется в числа с использованием таблицы base32. Первое число является версией свидетеля — `0`. Следующие числа, кроме последних шести, составляют программу свидетеля. Последние шесть чисел — это контрольная сумма.

❸ Производится проверка контрольной суммы; в данном примере ошибки отсутствуют.

❹ Программа свидетеля переписывается как последовательность 5-битных чисел.

❺ Биты переупорядочиваются в группы по 8 бит. Каждая такая группа представляет байт программы свидетеля.

❻ Эми извлекает программу свидетеля как `c8052b7...3176cba8`.

КОНТРОЛЬНАЯ СУММА

Я не буду углубляться в детали, связанные с контрольной суммой. Желающие могут обратиться непосредственно к тексту BIP173.

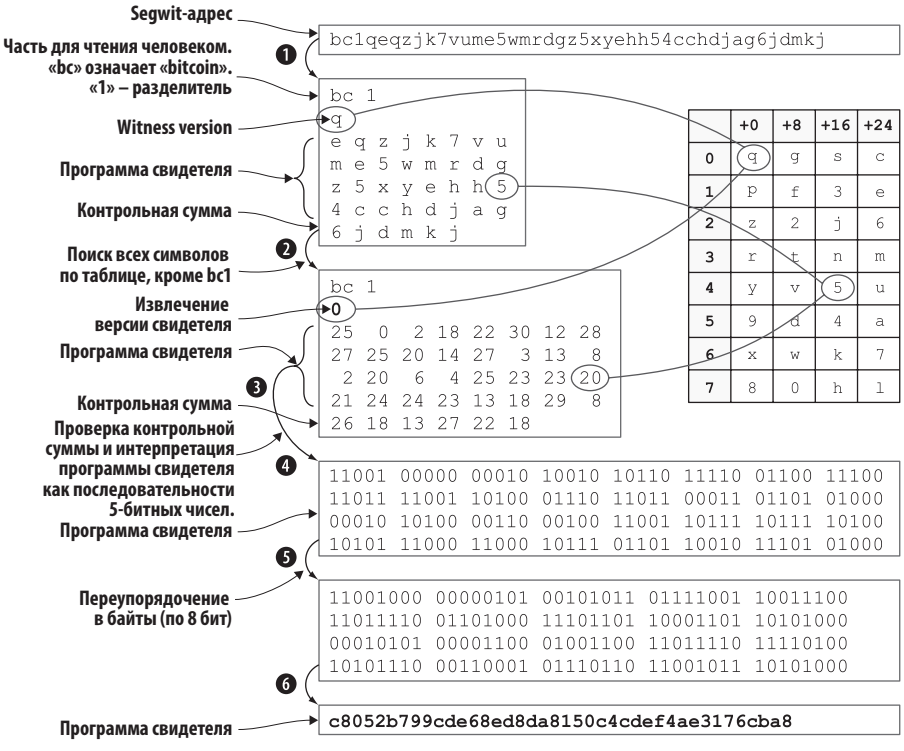


Рис. 10.12. Эми декодирует segwit-адрес и извлекает версию и программу свидетеля

Эми создает транзакцию с новым и необычным для нас типом сценария открытого ключа (рис. 10.13).

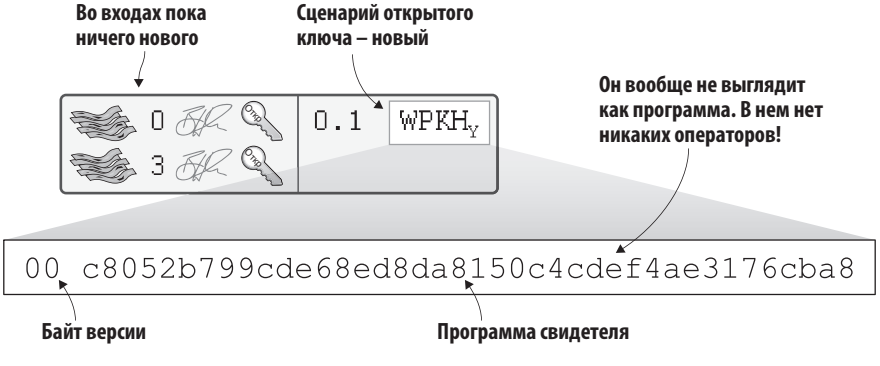


Рис. 10.13. Эми посылает 0.1 BTC на ваш segwit-адрес. Сценарий открытого ключа не содержит никаких операторов, только данные

Она посылает эту транзакцию в сеть Биткоин. Сеть примет транзакцию, потому что она правильно подписана старым способом. В конечном итоге она будет подтверждена и включена в блок. Ваш кошелек подтвердит получение денег, и вы отдадите ноутбук Эми.

Расходование segwit-выхода

Получив деньги, вы решаете потратить их на бывшую в употреблении машину для попкорна. Она стоит всего 0,09 BTC. Это почти даром! Предположим, что владелец попкорн-машины имеет segwit-адрес `bc1qlk34...ul0qwrqp`.

Ваша транзакция переводит деньги на segwit-адрес владельца машины для попкорна и платит 0,01 BTC за транзакцию (рис. 10.14). Вход имеет пустой сценарий подписи — подпись записывается в *поле witness* вложенной структуры свидетеля.

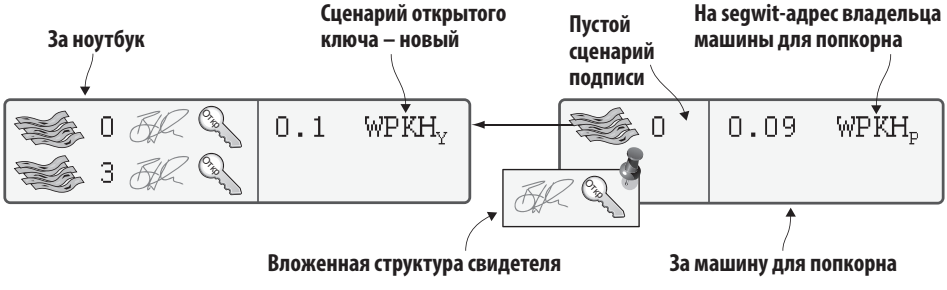


Рис. 10.14. Вы создаете транзакцию и посылаете деньги владельцу машины для попкорна

Если бы в этой транзакции было несколько входов, в структуре свидетеля было бы несколько полей witness, по одному для каждого входа. В одной транзакции можно одновременно использовать старые и segwit-входы. В этом случае поля witness, соответствующие входам в старом формате, будут пустыми, потому что их подписи будут находиться в соответствующих сценариях подписей, как обычно.

Проверка segwit-транзакции

Вы отправили транзакцию с оплатой за машину для попкорна в сеть Биткоин. Давайте посмотрим, как обновленный полный узел проверит эту транзакцию, прежде чем передать ее другим узлам (рис. 10.15). Поскольку узел

использует новейшее программное обеспечение, он знает, как обращаться с segwit-транзакциями.

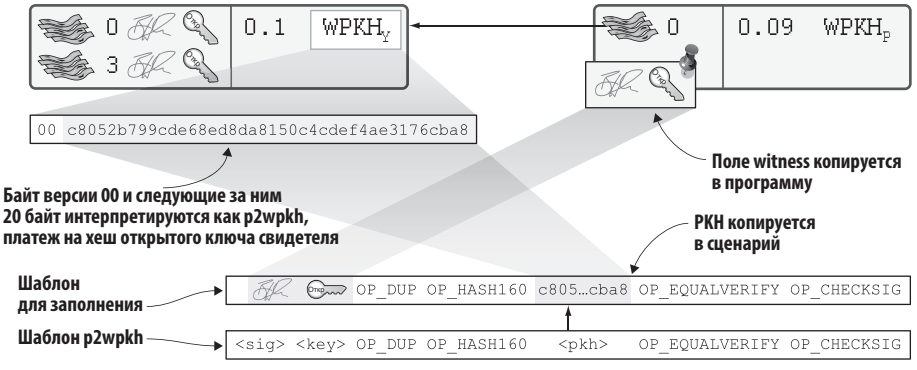


Рис. 10.15. Полный узел проверяет свидетеля вашей транзакции. Байт 00 и следующие за ним 20 байт интерпретируются особым образом

Полный узел с поддержкой segwit просматривает шаблон в сценарии открытого ключа, начиная с байта версии, за которым следует программа свидетеля, от 2 до 40 байт. В данном случае шаблон соответствует, то есть это segwit-выход.

На следующем шаге полный узел должен определить *тип,* или *версию*, этого segwit-выхода. На момент написания этих строк существовала только одна версия segwit-выхода: версия `00`. Эта версия имеет две разновидности:

ВСПОМНИТЕ P2SH

Segwit-выход идентифицируется по соответствию шаблону, в точности как выход p2sh, о котором говорилось в главе 5.

- *Платеж на хеш открытого ключа свидетеля* (pay-to-witness-public-key-hash, p2wpkh) определяется 20-байтной программой свидетеля, как в этом примере.
- *Платеж на хеш сценария свидетеля* (pay-to-witness-script-hash, p2wsh) определяется 32-байной программой свидетеля. Версию p2wsh мы рассмотрим далее в этой главе.

В этом случае у нас есть байт версии `00`, за которым следуют ровно 20 байт. Значит, это платеж p2wpkh. Если узел не опознает байт версии, он просто примет этот вход без дальнейшей обработки. Такое отношение к неизвестным номерам версий пригодится для будущих обновлений языка сценариев, поддерживающих прямую совместимость. Все segwit-узлы распознают версию `00`.

p2wpkh — самый простой из двух типов, потому что похож на хорошо известный p2pkh. Давайте посмотрим, как они работают:

- *p2pkh* — сценарий открытого ключа содержит фактический сценарий, который проверяет подпись в сценарии подписи;
- *p2wpkh* — фактический сценарий является предопределенным шаблоном, а программа свидетеля *является* хешем открытого ключа, который требуется вставить в шаблон. Подпись и открытый ключ извлекаются из структуры свидетеля.

ПОЧЕМУ «ПРОГРАММА СВИДЕТЕЛЯ»?

Название «программа свидетеля» выбрано потому, что ее можно рассматривать как программу на необычном языке. В версии `00` программа свидетеля — это единственный оператор, длина которого определяет поведение.

В результате получается одна и та же программа, которая обрабатывает оба эти типа. Разница лишь в том, откуда берутся компоненты. Однако между старыми и segwit-сценариями существуют и другие различия — например, значение `OP_CHECKSIG` изменилось, как будет показано в разделе «Новый метод хеширования подписей».

Зачем вообще использовать p2wpkh, если запускается та же программа-сценарий, что и в p2pkh? Напомню, что цель — устранить проблему пластичности транзакций. Это достигается удалением данных подписей из входов транзакции, чтобы никто не мог изменить ее идентификатор, внося небольшие изменения в сценарий подписи.

Полный узел проверил транзакцию и отправил своим соседям. Осталась только одна проблема: один из узлов не знает, что такое segwit. Это старый узел, который давно не обновлялся.

«Проверка» на старых узлах

Старый узел только что получил вашу транзакцию и должен ее проверить. Старые узлы ничего не знают о segwit или о свидетелях, приложенных к транзакциям. Старый узел загружает транзакцию как обычно, без сопровождающего ее свидетеля. На рис. 10.16 показано, что видит узел.

Поскольку узел не знает ничего другого, он создаст программу сценария, взяв пустой сценарий подписи и добавив сценарий открытого ключа. В результате получится программа:

```
00 c8052b799cde68ed8da8150c4cdef4ae3176cba8
```

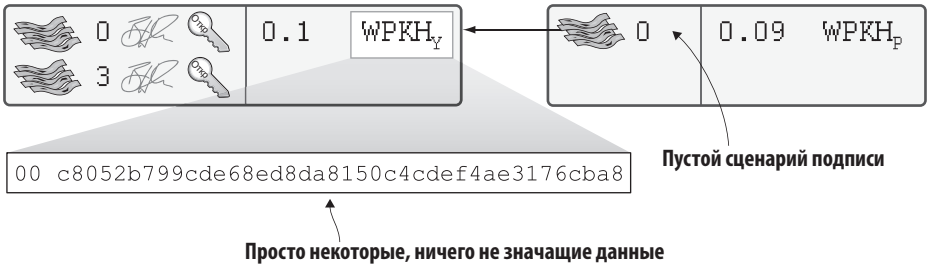


Рис. 10.16. Старый узел видит только два элемента данных в сценарии открытого ключа и пустой сценарий подписи

Узел выполнит эту программу. Программа поместит на стек два элемента данных — сначала `00`, а потом `c805...cba8`. После этого узлу останется только проверить истинность элемента `c805...cba8` на вершине стека. В Биткоин любое ненулевое значение считается истинным, поэтому сценарий завершится успехом и транзакция будет авторизована.

Такое поведение выглядит небезопасным. Оно известно как «*любой сможет потратить*», то есть любой сможет создать транзакцию, расходующую выход без обязательной подписи. Чтобы забрать деньги, достаточно создать вход с пустым сценарием подписи.

В главе 11 мы поговорим о том, как безопасно внедрять такие обновления, как segwit. А пока просто имейте в виду, что 95% хешрейта (майнеров) работает с поддержкой segwit. Если какая-то транзакция использует ваш выход на манер «любой сможет потратить» и майнер, использующий программное обеспечение, которое не поддерживает segwit, включит ее в блок, то этот блок будет отклонен 95% хешрейта и, соответственно, исключен из сильнейшей цепочки. Майнер потеряет награду за блок.

НЕСТАНДАРТНЫЕ ТРАНЗАКЦИИ

Обычно если узел не распознал тип использованного сценария, он перешлет транзакцию дальше. Такие транзакции считаются нестандартными. Эта политика в отношении пересылки снижает риск попадания в блок транзакций, использующих segwit-выход на манер «любой сможет потратить».

Включение segwit-транзакции в блок

Ваша segwit-транзакция распространилась по сети, и все узлы проверили ее. Теперь майнер собирается включить ее в новый блок. Предположим, что он использует современное программное обеспечение и поэтому знает о segwit. Давайте посмотрим, как он включит транзакцию в блок (рис. 10.17).

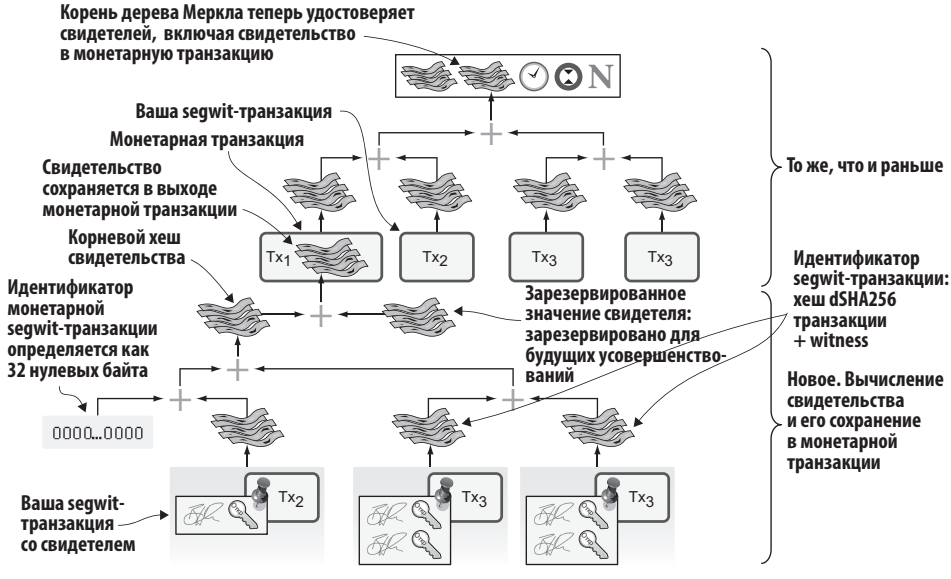


Рис. 10.17. Ваша segwit-транзакция включается в блок. Блок удостоверяет свидетельство, помещая свидетеля в выход монетарной транзакции

Блок конструируется так же, как и раньше, но с одним важным отличием. Решение segwit вводит новое правило: если в блоке есть segwit-транзакции, монетарная транзакция должна содержать выход со *свидетельством* (witness commitment), которое представляет собой объединенный хеш из *корневого хеша свидетельства* (witness root hash) и *зарезервированного значения свидетеля* (witness reserved value). Корневой хеш свидетельства — это корень Меркла *segwit-идентификаторов* (wtxids) всех транзакций в блоке. Segwit-идентификатор — это хеш транзакции *с ее свидетелем*, если он есть. Исключением является монетарная транзакция, чей segwit-идентификатор всегда определяется как 32 нулевых байта. Зарезервированное значение свидетеля предназначено для будущих обновлений системы.

Свидетельство записывается в выход `OP_RETURN` (рис. 10.18).

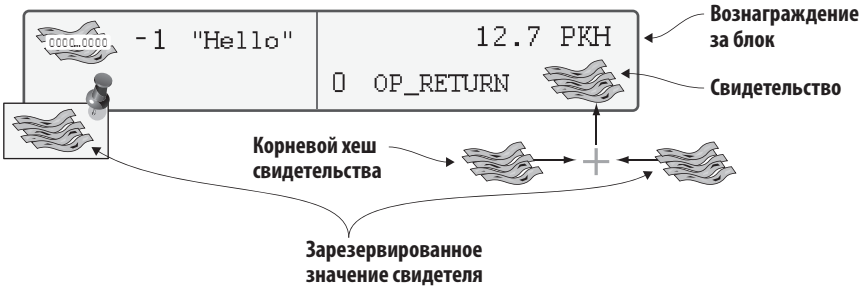


Рис. 10.18. Свидетель монетарной транзакции содержит зарезервированное значение свидетеля, а выход OP_RETURN — свидетельство

Зарезервированное значение свидетеля может быть любым значением. Но полному узлу, проверяющему этот блок, нужно как-то узнать это значение. Если узел не знает зарезервированного значения свидетеля, он не сможет вычислить свидетельство для сравнения со свидетельством в выходе `OP_RETURN`. Поэтому, чтобы полные узлы могли проверить свидетельство, зарезервированное значение свидетеля передается в свидетеле монетарной транзакции.

Проверка блока старым узлом

Новые полные узлы с поддержкой segwit будут считать блок на рис. 10.17 действительным, поэтому старые узлы, не поддерживающие решения segwit, тоже должны воспринимать его как действительный. Старый узел не будет загружать никаких свидетелей от своих соседей, потому что ничего не знает об их существовании (рис. 10.19).

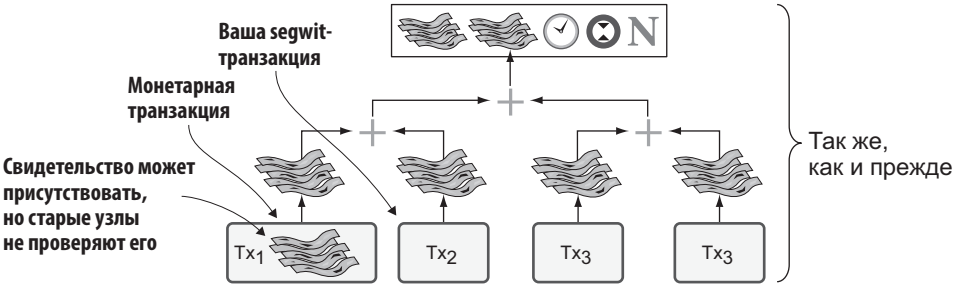


Рис. 10.19. Старый узел проверяет блок с вашей транзакцией. Он не проверяет подписи и свидетельства

Этот узел будет делать то, что и всегда — запускать сценарии из транзакций, которые будут выглядеть как выходы «любой сможет потратить».

Это нормально, двигаемся дальше. Если некоторые транзакции в блоке не являются segwit-транзакциями, они подвергнутся полной проверке.

Мы рассмотрели полный цикл распространения вашей транзакции на пути к владельцу машины для попкорна, который в конце концов передал ее вам.

Платеж на хеш сценария свидетеля

Давайте вспомним, о чем рассказывалось в разделе «Платеж на хеш сценария» в главе 5, где обсуждались платежи p2sh. Эти платежи перемещают часть сценария открытого ключа в расходующий вход. Посмотрим еще раз на благотворительный кошелек, который создали Джон, Элен и Фаиза (рис. 10.20).

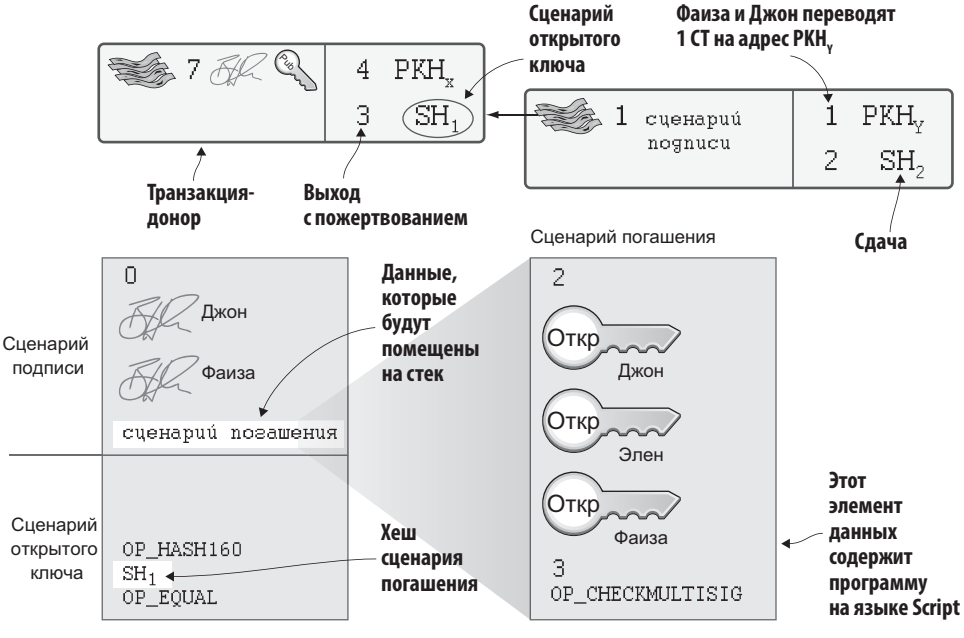


Рис. 10.20. Джон и Фаиза расходуют выход из своего кошелька, требующего несколько подписей

Идея заключалась в том, что плательщик — в данном случае донор — не должен платить более высокую комиссию за большой и сложный сценарий открытого ключа. Платить за сложность должен получатель, желающий использовать эту причудливую схему.

Решение segwit позволяет сделать то же самое, если оформить платеж на хеш сценария свидетеля (pay-to-witness-script-hash), который является segwit-версией для p2sh. И кто после этого скажет, что названия в Биткоин не фантастичны?

Предположим, что Джон, Элен и Фаиза используют segwit для своего благотворительного кошелька и предыдущий владелец машины для попкорна решил отдать на благотворительность деньги, вырученные от продажи.

Джон, Элен и Фаиза должны передать бывшему владельцу адрес p2wsh. Их *сценарий свидетеля* — это тот же *сценарий погашения*, который они использовали для получения платежей p2sh (рис. 10.21).

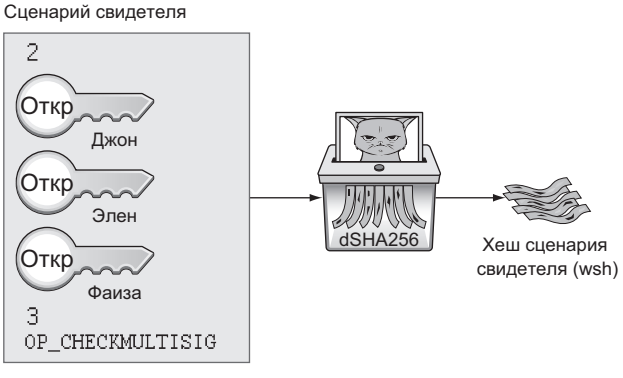


Рис. 10.21. Сценарий свидетеля хешируется в хеш сценария свидетеля

Они используют этот хеш сценария свидетеля для создания адреса p2wsh точно так же, как при создании адреса p2wpkh. Они кодируют

```
00 983b977f86b9bce124692e68904935f5e562c88226befb8575b4a51e29db9062
```

с использованием алгоритма Bech32 и получают адрес p2wsh:

```
bc1qnqaewluxhx7wzfrf9e5fqjf47hjk9jyzy6l0hpt4kjj3u2wmjp3qr3lft8
```

Этот адрес передается бывшему владельцу машины для попкорна, который создает и посылает транзакцию, подобную изображенной на рис. 10.22.

К транзакции прикладывается свидетель, точно так же как к вашей транзакции с переводом бывшему владельцу машины для попкорна. Единственная разница между этими транзакциями состоит в том, что их выходы имеют разную длину программы свидетеля. В вашей транзакции программа

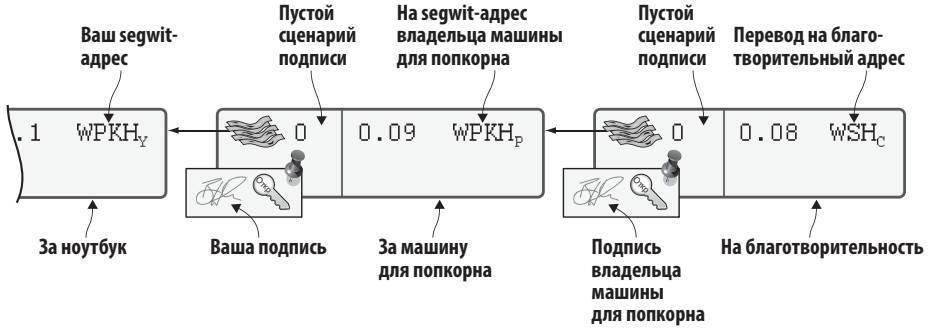


Рис. 10.22. Бывший владелец машины для попкорна переводит деньги на благотворительный адрес p2wsh

имела длину 20 байт, потому что это был хеш открытого ключа SHA256 + RIPEMD160, а программа свидетеля в транзакции бывшего владельца машины для попкорна имеет длину 32 байта, потому что это двойной хеш SHA256 сценария свидетеля.

Эта транзакция будет проверена и в итоге включена в блок.

Расходование транзакции p2wsh

Предположим, что Джон и Фаиза решили потратить 0,08 BTC, которые получили от бывшего владельца машины для попкорна, отправив их в приют для бездомных. У приюта тоже есть адрес p2wsh. Джон и Фаиза вместе создают транзакцию, изображенную на рис. 10.23.

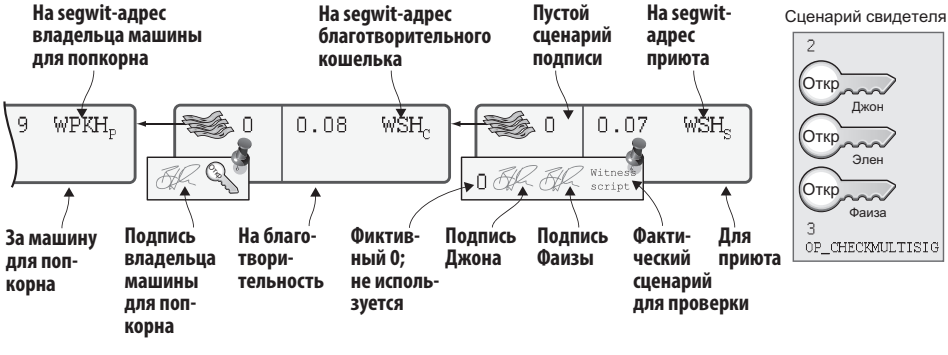


Рис. 10.23. Благотворительный перевод 0,07 BTC на адрес приюта. Свидетель здесь — это подписи, сопровождаемые элементом данных с фактическим сценарием свидетеля

Обратите внимание, что в сценарии подписи ничего нет. Когда мы использовали платежи p2sh в разделе «Платеж на хеш сценария» главы 5, сценарий подписи получился по-настоящему большим, потому что включал две подписи и сценарий погашения, который, в свою очередь, включал три открытых ключа. С внедрением решения segwit все данные теперь содержатся в свидетеле.

Проверка входа p2wsh

Чтобы проверить эту транзакцию, полный узел должен определить тип расходуемого выхода (рис. 10.24). Он просматривает выход, находит шаблон `<байт версии> <от 2 до 40 байт данных>` и приходит к выводу, что это segwit-выход. Следующее, что нужно проверить, — значение байта версии.

Байт версии содержит `00`. Версия `00` segwit-выхода может иметь программу свидетеля длиной 20 или 32 байта. Первая, как мы узнали в предыдущих разделах, выполняет платеж p2wpkh. В этом примере программа свидетеля имеет длину 32 байта, то есть это выход p2wsh.

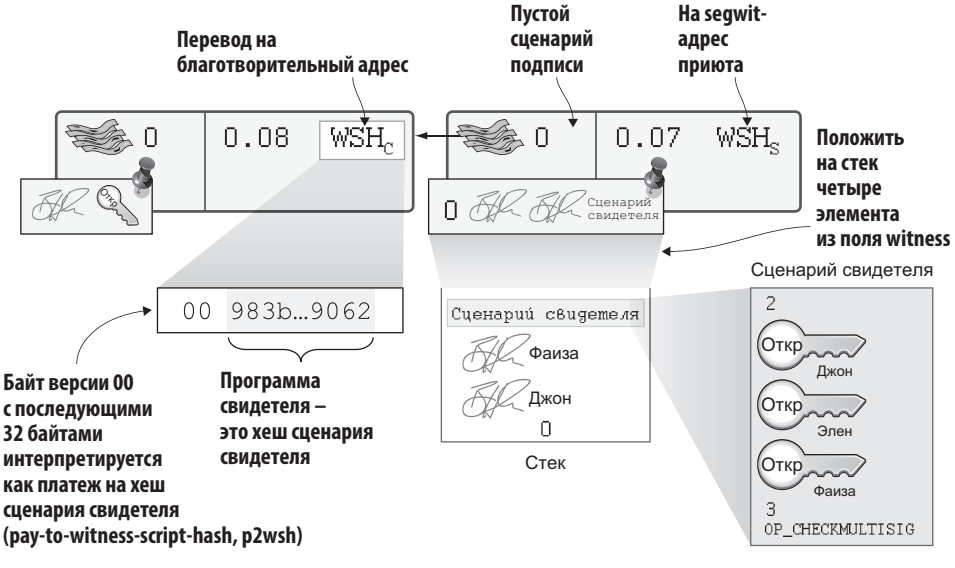


Рис. 10.24. Подготовка к проверке входа p2wsh

Когда расходуется выход p2wsh, применяются особые правила. Сначала элементы данных из поля witness расходующего входа помещаются на стек

программы. Затем верхний элемент на стеке, сценарий свидетеля, сверяется с программой свидетеля в выходе (рис. 10.25).

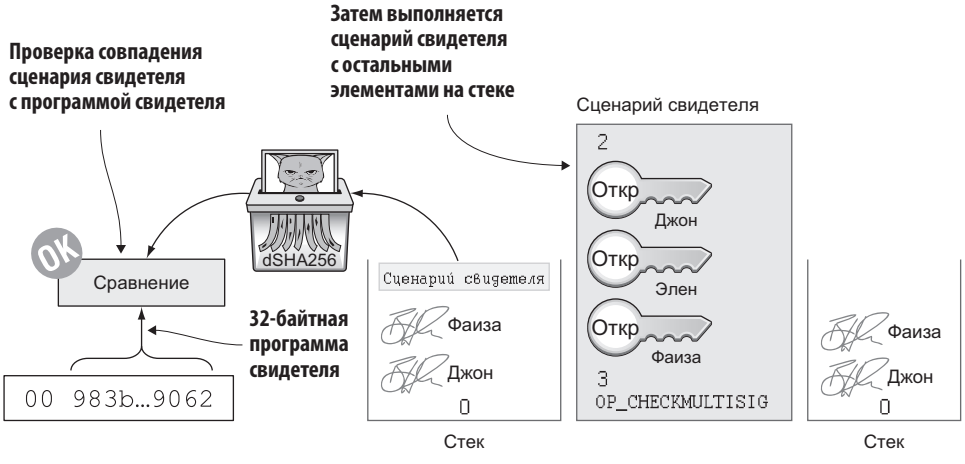


Рис. 10.25. Проверка поля witness из платежа p2wsh

Перед выполнением с тремя элементами на стеке сценарий свидетеля хешируется и сравнивается с программой свидетеля в расходуемом выходе. Этот процесс напоминает процесс проверки платежа p2sh.

Майнеры и полные узлы, проверяющие блоки, обрабатывают все segwit-транзакции одинаково, поэтому включение этой транзакции в блок ничем не отличается от включения транзакции p2wpkh.

Новый метод хеширования подписей

Одна из проблем, которую решает segwit, — неэффективное хеширование подписей. Как объясняется в разделе «Неэффективная проверка подписи», если количество входов удваивается, время, необходимое для проверки транзакции, увеличивается примерно в четыре раза. Это объясняется тем, что:

- удваивается количество проверяемых подписей;
- удваивается размер транзакции.

BIP143

Это решение описывается в BIP143, «Transaction Signature Verification for Version 0 Witness Program».

Если удвоить количество операций хеширования *и* удвоить объем данных, необходимых для обработки каждого хеша, общее время, затрачиваемое на хеширование, фактически увеличивается в четыре раза.

Решение заключается в создании подписей по шагам. Предположим, вы решили подписать все четыре входа транзакции, как показано на рис. 10.26.

ЭТО УПРОЩЕННОЕ ОПИСАНИЕ АЛГОРИТМА

В действительности создаются три разных промежуточных хеша: один для всех выходов, один для всех порядковых номеров и один для всех выходных точек. Однако суть остается прежней. Подробности ищите в BIP143.

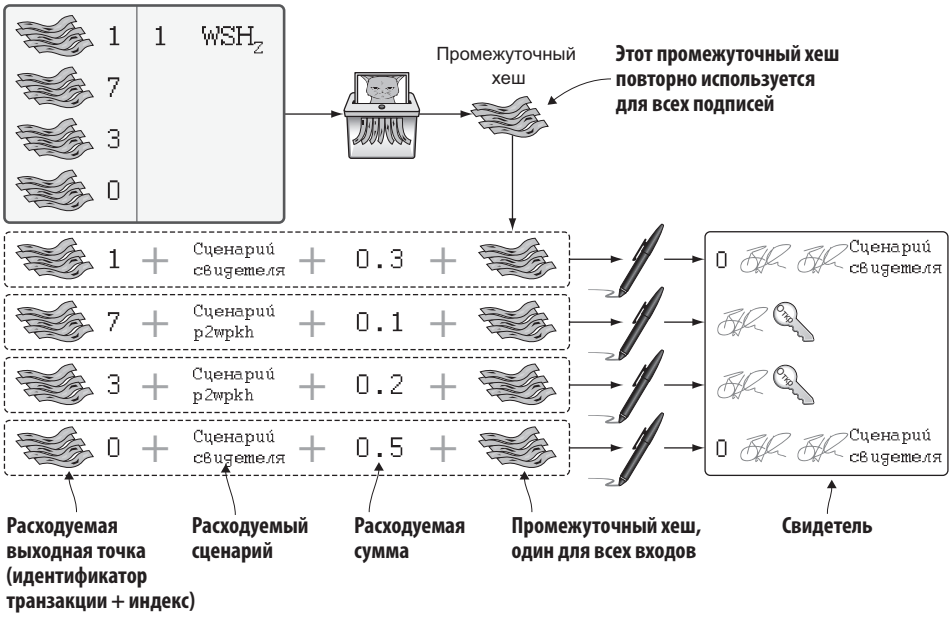
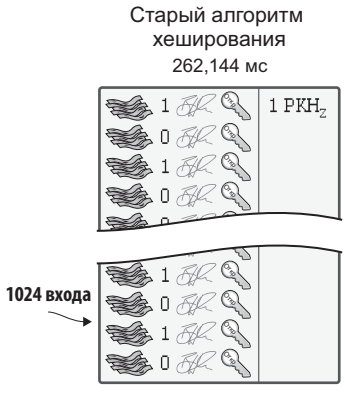


Рис. 10.26. Хеширование выполняется в два этапа. Для каждого входа повторно используется промежуточный хеш

Сначала создается промежуточный хеш всей транзакции. Если транзакция содержит не segwit-входы, соответствующие сценарии подписей очищаются перед хешированием. Промежуточный хеш удостоверяет все входы и выходы этой транзакции. Затем для каждого входа промежуточный хеш прибавляется к данным, соответствующим этому входу:

- *расходуемая выходная точка* (spent outpoint) — идентификатор транзакции и индекс выхода, который расходует данный вход;

- *расходуемый сценарий* (spent script) — сценарий свидетеля или сценарий p2wpkh, соответствующий расходуемому выходу;
- *расходуемая сумма* (spent amount) — сумма в BTC расходуемого выхода.



Основная часть транзакции хешируется только один раз для создания промежуточного хеша. Это резко уменьшает количество операций хеширования. Если количество входов удвоится, количество операций хеширования тоже удвоится. То есть время работы алгоритма находится в *линейной зависимости от количества входов, а не в квадратичной*. Время проверки транзакции с 1024 входами, изображенной на рис. 10.7, сокращается с 262 144 мс до 512 мс.

Подпись удостоверяет сумму

Почему в проверку включена расходуемая сумма? Старый алгоритм хеширования подписи не игнорировал ее. Это не имеет никакого отношения к увеличению эффективности хеширования, но устраняет еще одну проблему, с которой сталкиваются автономные и некоторые легкие кошельки.

АППАРАТНЫЕ КОШЕЛЬКИ

Аппаратный кошелек — это электронное устройство, предназначенное для защиты закрытых ключей. Неподписанные транзакции посылаются в устройство для подписи. Обычно, чтобы поставить подпись, такое устройство требует ввести пин-код.

Автономный кошелек, например аппаратный кошелек, не может знать, какая сумма расходуется. Если автономный кошелек используется для подписания транзакции, он не сможет показать пользователю сумму комиссии за транзакцию, потому что не видит расходуемую сумму (рис. 10.27). У него нет доступа к блокчейну.

Это верно как для обычных, так и для segwit-транзакций. Но в случае с segwit-транзакциями, когда подписи удостоверяют суммы расходуемых

выходов, кошелек должен откуда-то получить эти суммы, чтобы иметь возможность подписать транзакцию. Предположим, что суммы входов каким-то образом передаются в автономный кошелек вместе с транзакцией для подписания. Тогда кошелек может подписать транзакцию, используя эти суммы, и даже показать пользователю перед подписанием сумму комиссии.

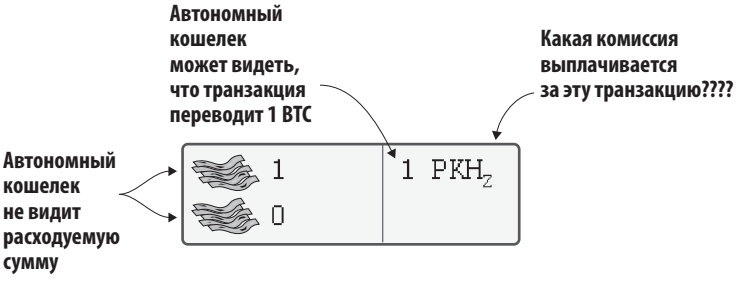


Рис. 10.27. Автономный кошелек не знает величину комиссии за транзакцию

Если автономный кошелек получит неправильную сумму, он не сможет определить это. Он не сможет проверить значения входов. Но поскольку теперь подписи удостоверяют суммы, транзакция окажется недействительной. Проверяющий узел будет знать правильные суммы и использовать их при проверке подписей. Проверка подписи завершится с ошибкой. Новый алгоритм хеширования подписи не позволяет обмануть кошелек и заставить его подписать действительную транзакцию с суммой, которую пользователь не намеревался платить.

Экономия пропускной способности

Решение segwit выносит подписи из транзакции, поэтому, когда легкий кошелек запрашивает транзакцию у полного узла, тот может отправить транзакцию без данных свидетеля. Это означает, что на передачу транзакции требуется меньше трафика. Этот факт можно использовать, чтобы:

- сохранить размер фильтра Блума и уменьшить объем трафика примерно на 50%;
- улучшить конфиденциальность за счет уменьшения размера фильтра Блума, чтобы получить больше маскирующих данных без увеличения трафика.

Расширение языка сценариев

Байт версии можно использовать для расширения языка сценариев в будущем. До появления решения segwit для внедрения новых возможностей в язык сценариев, таких как `OP_CSV`, можно было использовать только инструкции `OP_NOP`. Это не лучший подход, потому что:

- незанятые инструкции `OP_NOP` могут закончиться — их осталось всего восемь;
- инструкции `OP_NOP` нельзя переопределить произвольным образом; они все еще должны действовать как `OP_NOP` в случаях, когда новое поведение приводит к успеху.

Байт версии позволяет вводить в будущем более радикальные изменения. Нам доступны и небольшие изменения в поведении отдельных операторов, и реализация совершенно новых языковых конструкций.

Совместимость кошельков

Большинство старых кошельков не поддерживает отправку биткоинов на segwit-адреса. Обычно они поддерживают только адреса p2pkh и p2sh. Поэтому разработчики решения segwit создали *p2wsh, вложенный в p2sh*, и *p2wpkh, вложенный в p2sh*, — способы выполнить segwit-проверку вместо старой проверки сценария.

Предположим, у вас есть кошелек с поддержкой segwit и вы решили продать свою машину для попкорна своей соседке Нине. Но у Нины нет кошелька с поддержкой segwit. Она проводит платежи только по обычным адресам, таким как p2pkh и p2sh. Вы можете создать p2sh-адрес, на который Нина сможет перевести деньги (рис. 10.28).

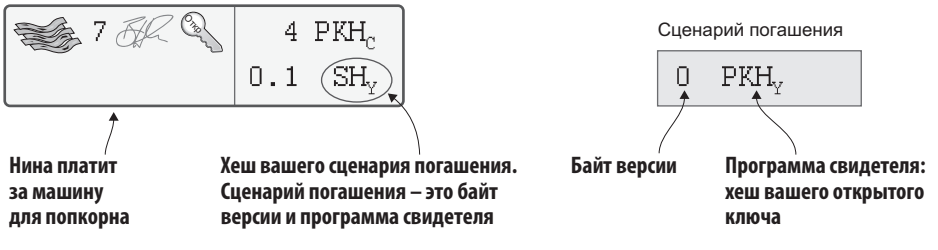


Рис. 10.28. Нина переводит 0,1 BTC на ваш segwit-кошелек, используя p2wpkh внутри адреса p2sh

Нина выполняет перевод на адрес `3KsJCgA6...k2G6C1Be` — старый адрес p2sh, который содержит хеш сценария погашения `00 bb4d4977...75ff02d1`. Этот сценарий, в свою очередь, состоит из байта версии `00`, за которым следует 20-байтная программа свидетеля. Это шаблон p2wpkh, с которым мы познакомились выше. Кошелек Нины ничего не знает об этом. Он видит только адрес p2sh и производит оплату на хеш этого сценария.

Позднее, когда вы решите потратить этот выход, вы создадите транзакцию, подобную той, что показана на рис. 10.29.

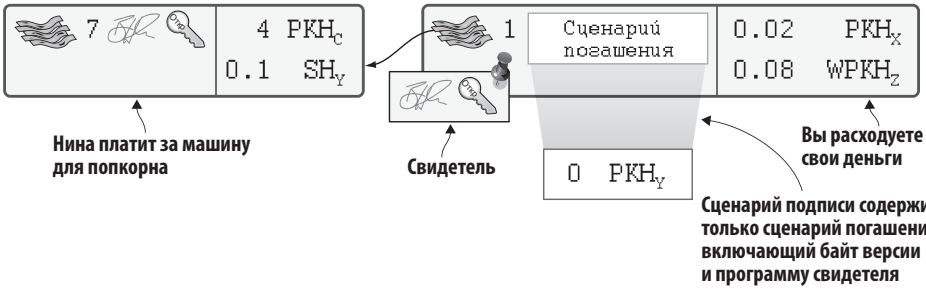


Рис. 10.29. Вы расходуете деньги, полученные от Нины, включив байт версии и программу свидетеля в сценарии погашения в сценарий подписи в своем входе

Вы создаете свидетеля так же, как для обычного входа p2wpkh, но дополнительно записываете сценарий погашения как отдельный элемент данных в сценарий подписи. Сценарий погашения включает байт версии, за которым следует 20-байтный хеш вашего открытого ключа. Используя этот сценарий подписи, старые узлы смогут проверить соответствие хеша сценария в расходуемом выходе хешу сценария погашения в сценарии подписи. Новые узлы определят, что сценарий погашения состоит из байта версии и программы свидетеля и соответствующим образом проверят свидетеля.

Этот способ вложения segwit-платежа в платеж p2sh также можно использовать для платежей p2wsh: вкладывая p2wsh в p2sh.

Еще раз о типах платежей

Мы познакомились с несколькими типами платежей. На рис. 10.30–10.35 показаны наиболее распространенные из них.

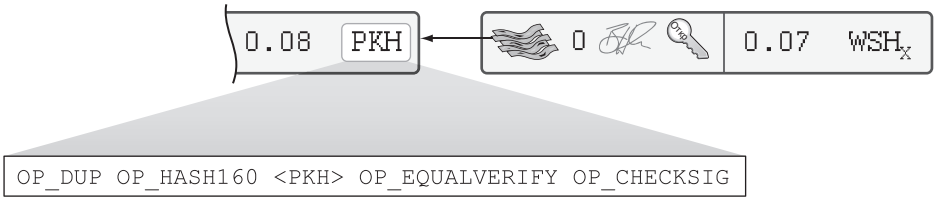


Рис. 10.30. p2pkh: адрес в формате 1<символы в кодировке base58>

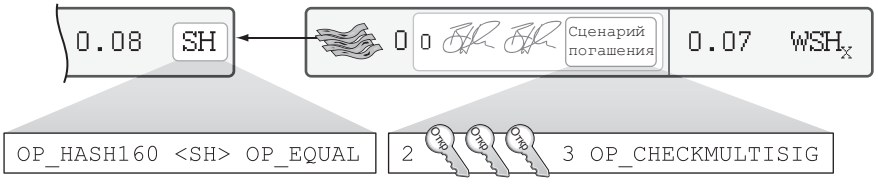


Рис. 10.31. p2sh: адрес в формате 3<символы в кодировке base58>

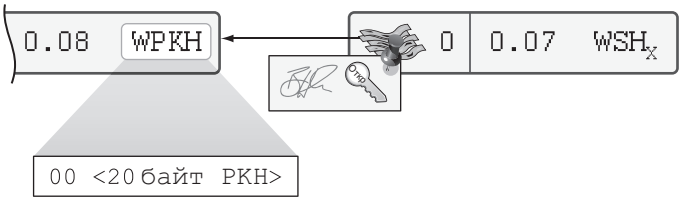


Рис. 10.32. p2wpkh: адрес в формате bc1q<38 символов в кодировке base32>

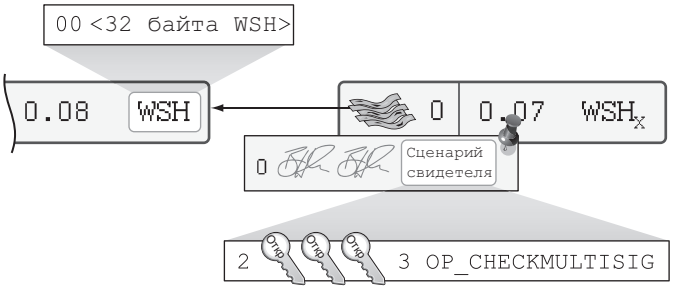


Рис. 10.33. p2wsh: адрес в формате bc1q<58 символов в кодировке base32>

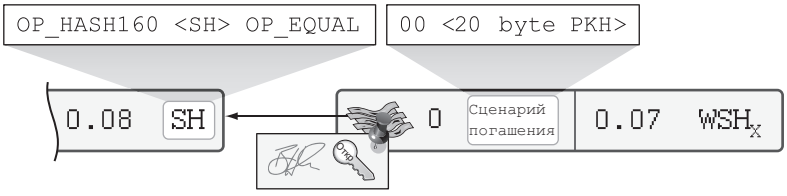


Рис. 10.34. p2wpkh, вложенный в p2sh: адрес в формате 3<символы в кодировке base58>

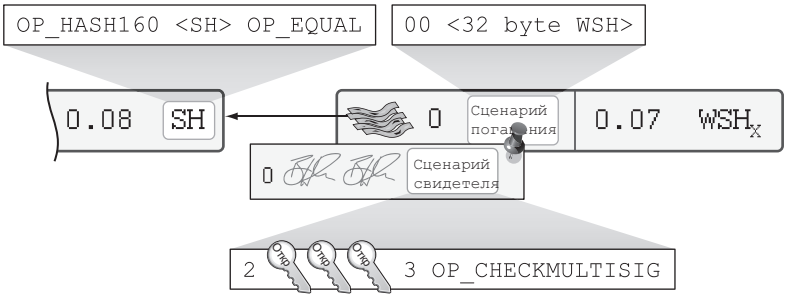


Рис. 10.35. p2wsh, вложенный в p2sh: адрес в формате 3<символы в кодировке base58>

Ограничения блоков

Блоки в Биткоин не могут иметь размер больше 1 000 000 байт и содержать больше 20 000 операций с подписями.

Ограничение размера блока

В 2010 году в программное обеспечение Биткоин было введено ограничение на размер блоков в 1 000 000 байт. Не совсем понятно, почему это было сделано, но большинство считает, что главной целью было ограничение возможностей для проведения атак типа «отказ в обслуживании» (Denial of Service, DoS). DoS-атаки нацелены на остановку или сбой узлов в сети Биткоин и нарушение нормального функционирования сети.

Один из способов нарушить работу сети — создать очень большой блок, который будет загружаться в течение 10 секунд даже при хорошем соединении с Интернетом. Это время может показаться не особенно большим,

но выгрузка такого блока пяти соседям займет 50 секунд. Это значительно замедлит распространение блока по сети и увеличит риск непреднамеренного расщепления блокчейна. Непреднамеренные расщепления разрешаются со временем, как вы видели в разделе «Выбор счастливых чисел» в главе 7, но в периоды таких расщеплений безопасность системы Биткоин в целом будет снижаться.

Другая потенциальная проблема, связанная с большими блоками, заключается в том, что они исключают из работы людей с медленным подключением к Интернету, потому что те не будут успевать за сетью, а также не обладающих необходимой вычислительной мощностью, объемом оперативной памяти или дискового пространства. Этим людям придется переключиться на системы с меньшей безопасностью, такие как легкие кошельки, что повлечет снижение безопасности всей сети.

Какими бы ни были причины, ограничение установлено.

Ограничение на количество операций с подписями

Ограничение на количество операций с подписями было установлено, потому что операции проверки подписей выполняются относительно долго, особенно в не segwit-транзакциях. Злоумышленник может заполнить транзакцию огромным количеством подписей, в результате чего проверяющие узлы окажутся занятыми проверками подписей в течение длительного времени. Предел в 20 000 таких операций на блок для предотвращения подобных атак выбран несколько произвольно.

Увеличение ограничений

Чтобы убрать или увеличить эти пределы, потребуется *хард-форк* (hard fork). Хард-форк — это такое изменение правил, которое приводит к тому, что старые и новые узлы расходятся во мнениях при выборе сильнейшей действительной цепочки. Подробнее о форках (ветвлениях) и обновлениях системы мы поговорим в главе 11. А пока предположим, что новые узлы считают вполне допустимыми блоки с размером 8 000 000 байт. Когда майнер попытается опубликовать блок размером больше 1 000 000 байт, новые узлы примут его, а старые — отвергнут. Произойдет постоянное расщепление блокчейна, и мы получим две разные криптовалюты.

Решение segwit дает возможность увеличить оба эти предела без хард-форка.

Увеличение максимального размера блока

Старое правило, ограничивающее размер блока 1 000 000 байт, продолжает действовать, поэтому старые узлы могут продолжать работать, как раньше. Новые узлы будут считать размер блока иначе, но совместимым способом. Байты в структуре свидетеля будут учитываться со «скидкой», в отличие от других байтов, таких как заголовок блока или выходы транзакции. В действие вводится новая мера — *вес блока*. Максимальный вес блока составляет 4 000 000 *весовых единиц* (weight units, WU; см. рис. 10.36).

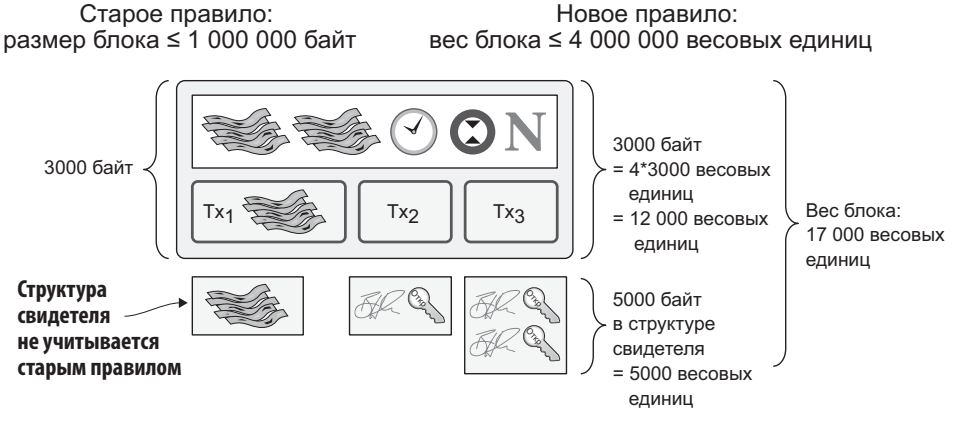


Рис. 10.36. Байты в структуре свидетеля и байты, не имеющие к ней отношения, учитываются по-разному. Байты в структуре свидетеля вносят меньший вклад в вес блока и вообще не включаются в традиционный размер блока

Будем далее называть блок без свидетеля базовым блоком:

- 1 байт базового блока эквивалентен 4 весовым единицам;
- 1 байт в структуре свидетеля эквивалентен 1 весовой единице.

В результате старое ограничение размера 1 000 000 байт остается, потому что в отношении базового блока новое и старое правила фактически совпадают. Но чем шире используется решение segwit, тем больше данных можно переместить из базового блока в структуру свидетеля и таким способом увеличить общий размер блока.

Пусть r — доля блока, занимаемая структурой свидетеля. Тогда, с учетом максимального веса блока 4 000 000, получаем общий размер блока T:

$$4(1-r)T + rT \le 4\times10^6$$
$$(4-3r)T \le 4\times10^6$$
$$T \le \frac{4\times10^6}{4-3r}.$$

Подставляя разные значения r в эту формулу, получаем разный максимальный общий размер блока, как показано в табл. 10.2.

Таблица 10.2. Максимальный размер блока для разных долей блока, занимаемых структурой свидетеля

r (байт в структуре свидетеля / всего байт)	Максимальный общий размер блока (байт)
0	1 000 000
0.1	1 081 081
0.3	1 290 323
0.5	1 600 000
0.6	1 818 182
0.7	2 105 263
0.8	2 500 000

По мере увеличения доли свидетельских данных можно увеличивать количество транзакций. Как следствие — увеличивается максимальный размер блока.

Меньший вес данных в структуре свидетеля реализован по нескольким причинам:

- Сценарии подписи и свидетельства не входят в набор UTXO. Данные, поступающие в набор UTXO, имеют более высокую стоимость, поскольку набор UTXO предпочтительнее хранить в ОЗУ для быстрой проверки транзакции.
- Это дает биржам и разработчикам кошельков и умных контрактов больше стимулов создавать меньше выходов, что способствует уменьшению размера набора UTXO. Например, биржа может объединить несколько выходов в один.
- Структура свидетеля не посылается в легкий кошелек.

Увеличение максимального числа операций с подписями

Вместе с увеличением максимального размера блока решение segwit должно также увеличивать максимальное количество операций с подписями; увеличение количества транзакций в блоке означает, что для их обработки требуется выполнить большее количество операций с подписями. Увеличить это ограничение можно точно так же, как мы увеличили максимальный размер блока.

Увеличим число операций с подписями с 20 000 до 80 000 и будем считать, что для обработки каждой устаревшей подписи требуется четыре операции, а для каждой segwit-подписи — одна операция. Обработка segwit-подписи обходится дешевле устаревшей, потому что выполняется эффективнее, как обсуждалось в разделе «Новый метод хеширования подписей».

В результате получится то же, что и в случае с увеличением максимального размера блока. Если блок содержит только устаревшие входы, продолжит действовать лимит в 20 000 операций. Если блок содержит только segwit-входы, в действие вступит новый лимит в 80 000 операций. Любая комбинация устаревших и segwit-входов в блоке даст в результате ограничение где-то между 20 000 и 80 000 операций с подписями.

Повторение

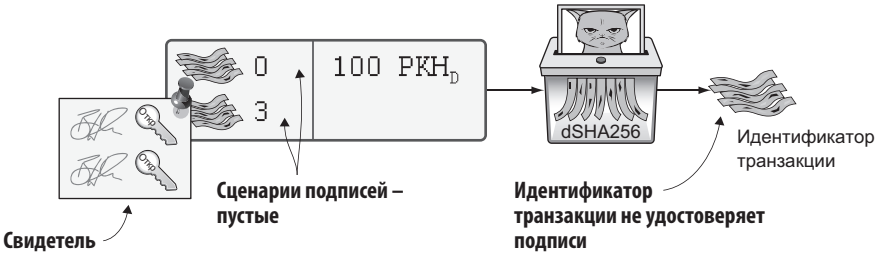
В этой главе мы рассмотрели решение под названием «Отдельный свидетель» (segregated witness), устраняющее несколько проблем:

- *Пластичность транзакций* — когда идентификатор транзакции может измениться без изменения эффекта, который оказывает транзакция. Такое изменение может привести к разрыву ссылок между транзакциями и сделать дочернюю транзакцию недействительной.
- *Неэффективная проверка подписи* — когда число входов в транзакции удваивается, время проверки транзакции увеличивается в четыре раза. Это связано с тем, что размер транзакции и количество проверяемых подписей удваиваются.
- *Потеря пропускной способности* — для проверки доказательства Меркла легкие кошельки должны загружать транзакции целиком, включая все подписи, но подписи для них бесполезны из-за недоступности проверки потраченных выходов.

- *Сложность обновления* — язык сценариев допускает обновление в весьма ограниченных пределах. Осталось всего несколько инструкций `OP_NOP`, которые, кстати, нельзя переопределить как заблагорассудится. В случаях, когда новое поведение оператора предполагает успех, он должен вести себя точно так же, как `OP_NOP`.

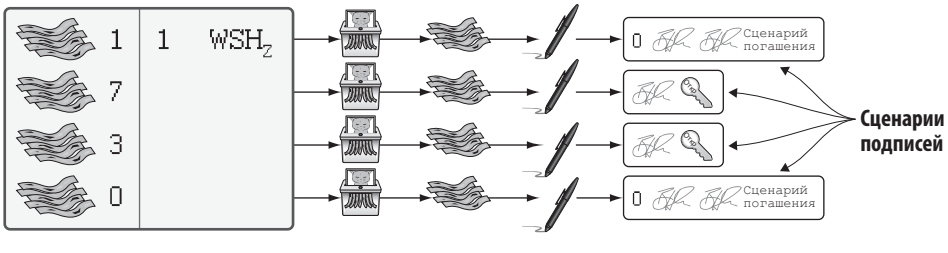
Решения

Благодаря перемещению подписей из транзакции в отдельную структуру они не учитываются при вычислении идентификатора транзакции.

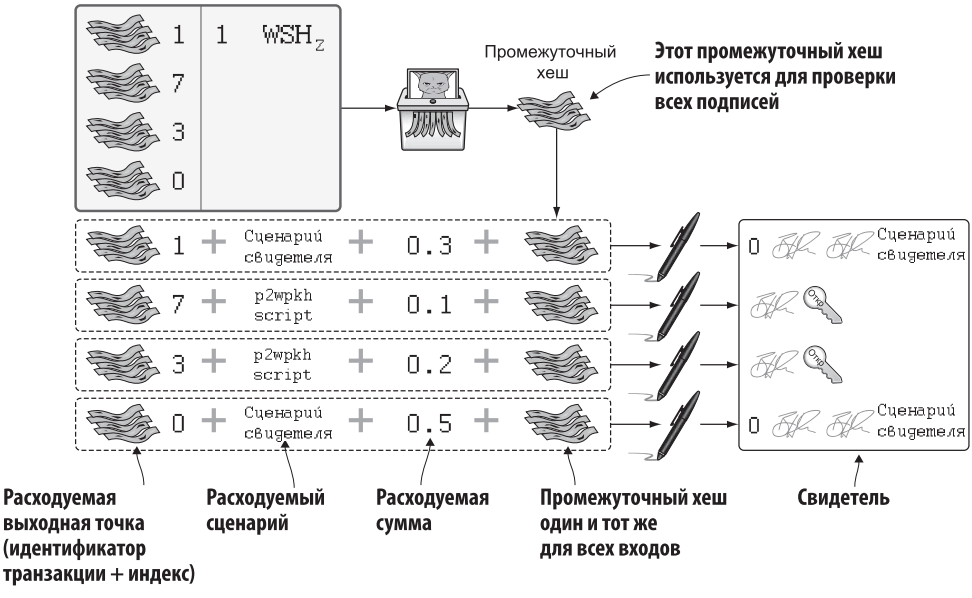


Если подпись окажется недействительной, это не повлияет на идентификатор транзакции. Неподтвержденные цепочки транзакций становятся неразрывными.

С новым алгоритмом хеширования подписей время проверки увеличивается *линейно* с увеличением числа входов. Старый алгоритм хеширует всю транзакцию для каждой подписи.



После перемещения подписей в структуру свидетеля транзакция хешируется только один раз.

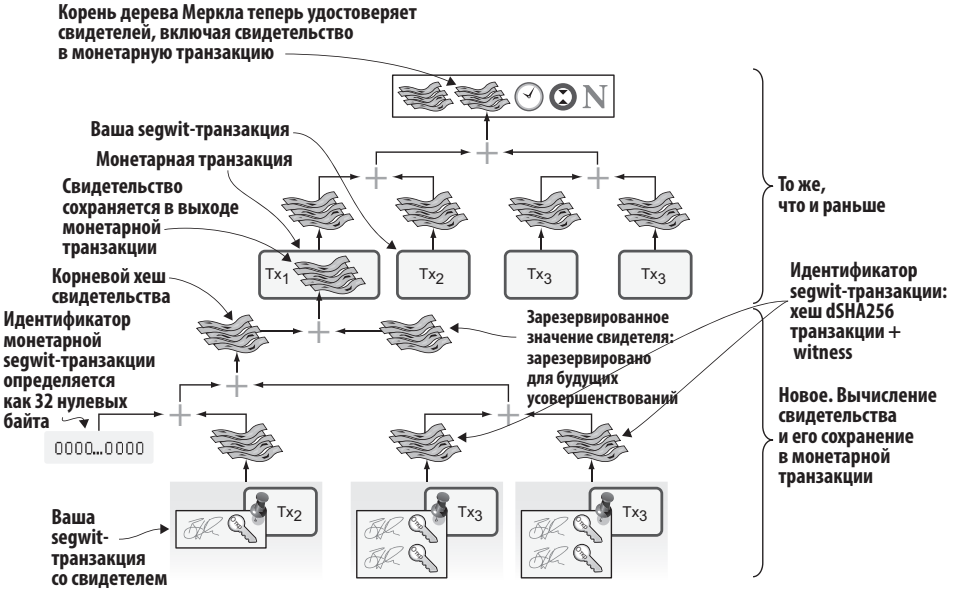


Промежуточный хеш повторно используется для проверки каждой подписи, что значительно уменьшает общее количество операций хеширования.

Требования к пропускной способности легких кошельков уменьшаются, так как им не нужно загружать структуру свидетеля, чтобы убедиться, что транзакция включена в блок. Они могут использовать полученную экономию для повышения конфиденциальности, уменьшая размер фильтра Блума, или уменьшать объем пересылаемых данных без ущерба для конфиденциальности.

Версия свидетеля в сценарии открытого ключа позволяет в будущем расширять язык сценариев. При этом расширения и изменения могут быть произвольно сложными — никаких ограничений на функциональность не накладывается.

Новые правила применяются к блокам, содержащим segwit-транзакции. Выход в монетарной транзакции должен удостоверяться всеми свидетелями блока.



Старые узлы смогут работать как прежде, потому что не знают о свидетельстве в монетарной транзакции. Это позволяет внедрять решение segwit, не разрушая блокчейн и не расщепляя его на две отдельные криптовалюты.

Упражнения

Для разминки

10.1. Какая часть транзакции является причиной пластичности?

10.2. Почему пластичность транзакций расценивается как проблема?

10.3. Почему время проверки старых транзакций увеличивается в квадратичной зависимости от увеличения числа входов?

10.4. Почему легким кошелькам нужны подписи в старых транзакциях, чтобы убедиться, что они включены в блок?

10.5. Предположим, вы решили добавить новую инструкцию в язык сценариев Биткоин и переопределить поведение `OP_NOP5`. О чем следует помнить при разработке нового поведения, чтобы избежать расщепления блокчейна (из-за того что не все узлы обновятся одновременно)?

10.6. Какие из следующих адресов являются segwit-адресами? К каким видам segwit-адресов они относятся?

а) `bc1qeqzjk7vume5wmrdgz5xyehh54cchdjag6jdmkj`

б) `c8052b799cde68ed8da8150c4cdef4ae3176cba8`

в) `bc1qnqaewluxhx7wzfrf9e5fqjf47hjk9jyzy6l0hpt4kjj3u2wmjp3qr3lft8`

г) `3KsJCgA6ubxgmmzvZaQYR485tsk2G6C1Be`

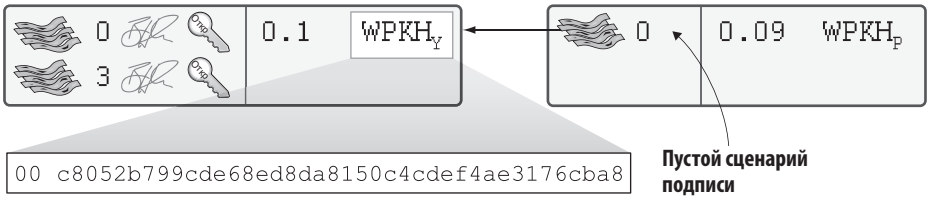
д) `00 bb4d49777d981096a75215ccdba8dc8675ff02d1`

10.7. Для чего используется версия свидетеля? Версия свидетеля — это первое число в segwit-выходе, например `00` в

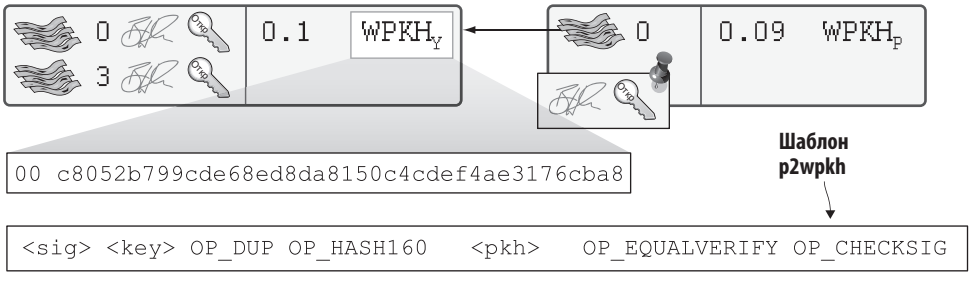
```
00 bb4d49777d981096a75215ccdba8dc8675ff02d1
```

Придется пораскинуть мозгами

10.8. Объясните, как старый узел определит действительность segwit-транзакции, которая ничего не знает о segwit. Вот что видит старый узел:

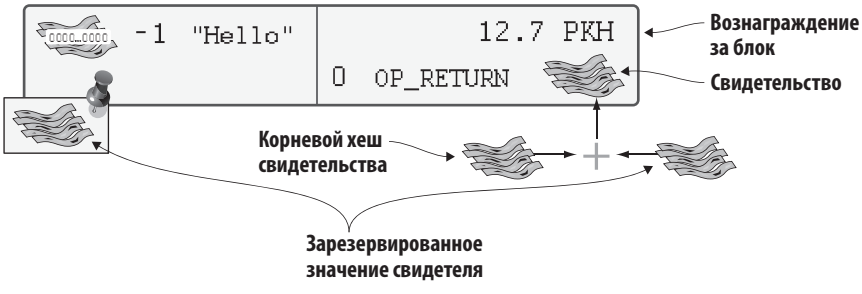


10.9. Объясните, как новый узел проверит segwit-транзакцию, которая ничего знает о segwit. Вот что видит новый узел:



10.10. Предположим, вы решили обновить систему Биткоин так, чтобы свидетельство тоже удостоверяло комиссионные за транзакции в блоке

в дополнение к корневому хешу свидетеля, путем создания дерева Меркла со всеми комиссионными отчислениями. Подумайте, как дерево Меркла могло бы удостоверить все комиссионные отчисления так, чтобы не нарушить совместимость со старыми узлами. Вам не нужно задумываться об изменении этого усовершенствования в будущем, потому что это слишком сложно. Используйте следующий рисунок в качестве подсказки:



10.11. Как старые и новые узлы будут проверять блоки, содержащие удостоверение из предыдущего упражнения?

Итоги

- Решение segwit перемещает сценарии подписей из транзакций, чтобы решить проблему пластичности транзакций.
- Решение segwit использует новый алгоритм хеширования подписей, ускоряющий проверку транзакций. Это помогает узлам затрачивать меньше вычислительных ресурсов.
- Благодаря экономии трафика из-за отсутствия необходимости загружать свидетелей, легкие кошельки способны улучшить конфиденциальность.
- Байт версии в сценарии открытого ключа упрощает внесение изменений в язык сценариев.
- Подсчитывая байты в структуре свидетеля со скидкой, можно несколько увеличить максимальный размер блока.
- Новый формат адреса помогает кошелькам отличать устаревшие платежи от segwit-платежей.
- Segwit-адреса можно встраивать в адреса p2sh, чтобы старые кошельки могли отправлять деньги в segwit-кошельки.

11 Апгрейды Биткоин

Эта глава охватывает следующие темы:

- ✓ хард- и софт-форк;
- ✓ безопасное совершенствование Биткоин;
- ✓ как пользователи устанавливают правила.

Для успешного понимания идей, описываемых в этой главе, необходимо четко понимать, что такое блокчейн (глава 6), доказательство работы (глава 7) и одноранговая сеть (глава 8). Если у вас возникли сложности при чтении этих глав, попробуйте еще раз перечитать их, прежде чем продолжить. Но, конечно, вы можете просто продолжить чтение дальше.

В Биткоин есть два пути изменения правил согласования: путем софт- или хард-форка. Эти два пути изменения правил имеют принципиальные отличия. С этими отличиями вы познакомитесь в разделе «Форки в Биткоин» и там же узнаете, что происходит, когда разные узлы используют разные правила согласования. Важно понять эти отличия, прежде чем приступать к изучению способов безопасного совершенствования правил согласования в Биткоин.

Внедрение изменений в правила согласования в сети Биткоин может быть затруднено. Каждый узел в Биткоин является суверенным, и никто не может диктовать, каким программным обеспечением должны пользоваться люди, — решение принимают сами пользователи. Это затрудняет внедрение, или *развертывание*, измененных правил согласования без широкой под-

держки пользователей и майнеров. *Механизмы развертывания* развивались со временем, и мы посмотрим, как это происходило, и познакомимся с текущим состоянием механизмов развертывания.

На момент написания этих строк большинство (некритичных) изменений в правилах согласования было сделано через *софт-форки, активируемые майнерами* (miner-activated soft forks), когда майнеры сигнализируют о поддержке новых правил и в итоге начинают их применять. Но при таком подходе возникают некоторые проблемы — например, крупный майнер может наложить вето на изменение, несмотря на широкую поддержку пользователями. Решение этой проблемы было найдено в *софт-форках, активируемых пользователями* (user-activated soft forks). Это означает, что власть принадлежит большинству: люди, использующие Биткоин, составляют *экономическое большинство*. Именно экономическое большинство и определяет правила, и это понимание реализуется на практике с помощью софт-форков, активируемых пользователями.

Форки в Биткоин

Программное обеспечение с открытым исходным кодом — это программное обеспечение, которое можно свободно загружать, использовать, проверять, изменять и распространять по своему усмотрению. Значительная часть программного обеспечения, которое вы используете ежедневно, распространяется с открытым исходным кодом. Может быть, вы используете веб-браузер Google Chrome или мобильный телефон с операционной системой Android. Все это примеры программного обеспечения с открытым исходным кодом.

РАЗНЫЕ ОПРЕДЕЛЕНИЯ

Люди определяют термин *ветвление* по-разному. В этой книге я использую определение, которое мне кажется наилучшим, а именно «изменение правил согласования».

Проекты с открытым исходным кодом могут *ветвиться*. Скопировав исходный код Linux, внеся в него некоторые изменения и занявшись распространением измененной версии, вы создадите свою *ветвь* (форк, fork) проекта.

Биткоин — это проект с открытым исходным кодом, который может ветвиться, как и любой другой проект с открытым исходным кодом, такой как Linux. Но в этой книге словом *ветвление* мы будем обозначать нечто иное.

В контексте Биткоин термин «ветвление» (форк) означает изменение правил согласования. Правила согласования определяют, что такое действительный блокчейн. Когда некоторый набор узлов использует одни и те же правила согласования, между ними возникает консенсус (согласие) относительно того, что представляет собой текущий набор неизрасходованных выходов транзакций (Unspent Transaction Outputs, UTXO), то есть «кто чем владеет». Проще говоря, ветвление изменяет определение действительного блокчейна.

Например, правило, которое ограничивает вес блока 4 000 000 весовых единиц, является правилом согласования. Изменение этого ограничения привело бы к ветвлению. Но политика пересылки, которая мешает пересылке транзакций с небольшими комиссионными, не является частью правил согласования. Изменение этой политики не влечет за собой ветвление.

Вы можете изменить правила согласования в Bitcoin Core, в скопированной версии Bitcoin Core или в любой другой программе поддержки сети Биткоин. Если кто-то начнет использовать вашу модифицированную программу, он создаст ветвление.

В общем случае ветвления в Биткоин классифицируются следующим образом (рис. 11.1):

- *Хард-форк* — вызывается ослаблением правил согласования. Некоторые блоки, недействительные для узлов, использующих старую версию программы, будут считаться действительными для узлов, использующих новую версию. Удвоение максимально допустимого веса блока — это жесткое ветвление.
- *Софт-форк* — вызывается ужесточением правил согласования. Все блоки, действительные для узлов, использующих новую версию программы, будут считаться действительными для узлов, использующих старую версию. Но некоторые блоки, считающиеся действительными для узлов, использующих старую версию программы, будут считаться недействительными для узлов, использующих новую версию. Уменьшение максимально допустимого веса блока — это софт-форк.

Изменения, которые не меняют правил согласования, такие как изменение цвета графического интерфейса пользователя или добавление новой функции в протокол одноранговой сети, не являются ветвлениями в Биткоин. Но их можно считать ветвлениями проекта программного обеспечения в традиционном смысле. С этого момента я буду использовать термин *форк* только для обозначения изменений в правилах согласования.

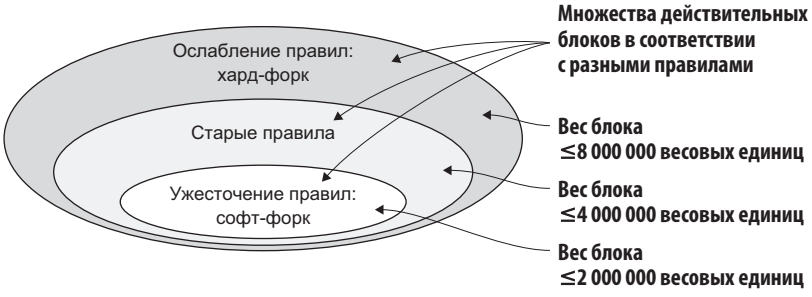
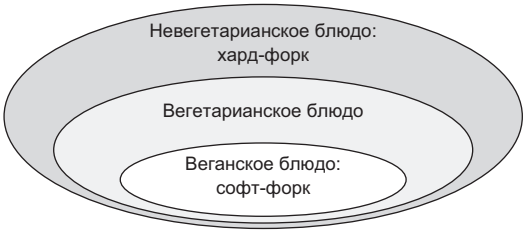


Рис. 11.1. Софт-форк ужесточает правила согласования, тогда как хард-форк ослабляет их — например, уменьшение и увеличение максимального веса блока соответственно

Чтобы провести аналогию с форками, представьте популярный вегетарианский ресторан, куда ходят пообедать многие вегетарианцы. В этом ресторане подают только вегетарианские блюда. Ресторан — это майнер, его гости — полные узлы, а блюда, которые подают в этом ресторане, — блоки. Ресторан готовит блюда, которые едят гости, — майнер производит блоки, которые принимают полные узлы.



Теперь представьте, что ресторан изменил свое меню, как показано в табл. 11.1.

Создав форк (софт или хард), вы рискуете расщепить блокчейн, если кто-нибудь начнет использовать вашу версию программы. Одни узлы будут следовать за самой сильной цепочкой, которая действительна согласно старым правилам, а другие узлы, на которых используется ваше программное обеспечение, будут следовать за самой сильной цепочкой, которая действительна согласно вашим новым правилам. В результате может произойти расщепление блокчейна.

Далее мы рассмотрим несколько примеров, демонстрирующих происходящее в разных сценариях. Начнем с самого простого случая: с изменения,

которое не влияет на правила согласования. Названием *старый Биткоин* мы будем обозначать предыдущую версию программы, а названием *новый Биткоин* — измененную версию. Узел, на котором выполняется старый Биткоин, мы будем называть *старым узлом*, а узел, на котором выполняется новый Биткоин, — *новым узлом*. Данные — например, блок, — созданный новым узлом, будем называть *новым блоком*. Аналогично транзакцию, созданную старым узлом, будем называть *старой транзакцией*.

Таблица 11.1. Ресторан может произвести хард-форк, добавив мясные блюда в свое меню, или софт-форк, ограничив меню веганскими блюдами

В вегетарианском ресторане подаются...	Примут это блюдо гости?	Тип форка	Почему
Вегетарианские блюда	Да	Нет форка	Вегетарианцы получают свои вегетарианские блюда
Невегетарианские блюда	Нет	Хард-форк	Правила *ослаблены*. Вегетарианцы больше не могут обедать здесь
Веганские блюда	Да	Софт-форк	Правила *ужесточены*. Вегетарианцы по-прежнему могут приходить сюда пообедать

Изменения, не касающиеся правил согласования

Предположим, вы решили добавить новую «функцию» в сетевой код Bitcoin Core — новый тип сетевого сообщения с именем `kill`, которое один узел Биткоин сможет отправить другому узлу. Узел, получивший это сообщение, должен немедленно отключиться. Только новые узлы будут знать, как реагировать на входящее сообщение `kill`. Старые узлы будут игнорировать неизвестное им сообщение (рис. 11.2).

Большинство посчитает ваши изменения огромным риском для безопасности. Они не хотят, чтобы их узлы мог останавливать случайный незнакомец из Интернета. Вам будет сложно убедить их использовать новый Биткоин. Вы не сможете навязать это программное обеспечение кому-либо; люди должны сами активно хотеть установить его, чтобы новый Биткоин получил широкое распространение в сети.

Глупые изменения, такие как сообщение `kill`, не сделают программное обеспечение популярным в мире открытого исходного кода.

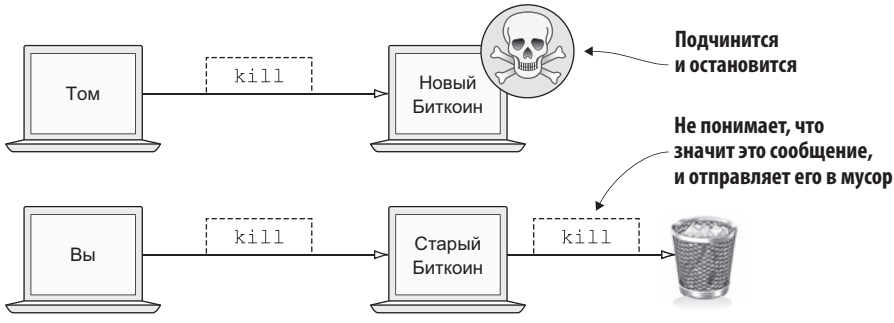


Рис. 11.2. Новое сообщение будет приниматься новыми и игнорироваться старыми узлами

Попробуем добавить что-нибудь более полезное

Предположим, вы изобрели что-то действительно очень полезное, пусть это будут *особо компактные блоки*. Компактные блоки позволяют одноранговому узлу отправить блок другому одноранговому узлу, не отправляя полный блок. Допустим, что этот метод опирается на тот факт, что узел-получатель уже получил большинство транзакций, включенных в блок. Напомню, что транзакции сначала рассылаются в сети поодиночке, а затем повторно рассылаются, но уже в составе блока, после подтверждения транзакции.

BIP152

Такое решение было реализовано в Bitcoin Core в 2016 году и значительно ускорило распространение блоков в сети Биткоин. Этот метод подробно описывается в BIP152, «Compact Block Relay». Здесь я приведу упрощенное описание.

Рашид отправляет блок узлу Ци (рис. 11.3), и было бы здорово исключить из блока транзакции, которые у Ци и так есть. Требования к пропускной способности резко снизились бы.

Теперь Рашид может отправить только заголовок блока и список идентификаторов транзакций (рис. 11.4). Получив заголовок, Ци может восстановить блок, собрав его из транзакций, которые у нее уже имеются, и из сообщения Рашида. Если у Ци не окажется какой-то транзакции, она сможет запросить ее у Рашида.

Согласно протоколу, обмен начинается с того, что Рашид отправляет Ци сообщение `cmpctblock`. Ци использует это сообщение для воссоздания блока и включения в него транзакций, которые у нее уже имеются. Если ей это удастся, она сможет начать проверять блок. Если какие-то транзакции отсутствуют у нее, она запросит их у Рашида, послав сообщение `getblocktxn`

со списком индексов недостающих транзакций. В ответ Рашид вернет сообщение `blocktxn`, содержащее запрошенные транзакции.

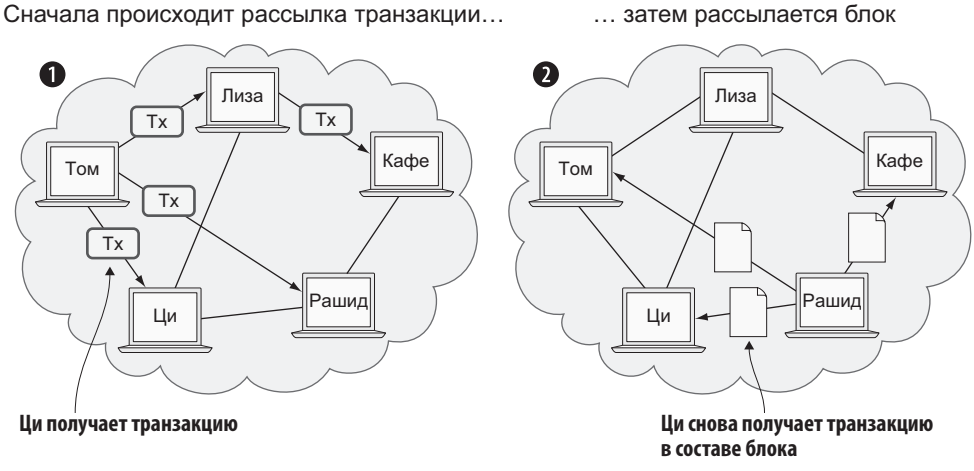


Рис. 11.3. Ци получает транзакцию дважды: сначала в процессе рассылки транзакции, а затем в процессе рассылки блока

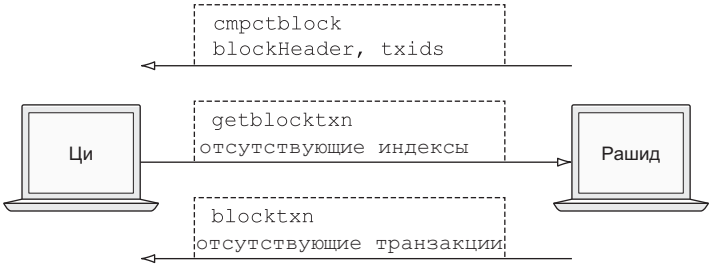


Рис. 11.4. Поддержка компактных блоков в действии.
Рашид посылает Ци только действительно необходимые данные

Имейте в виду, что это упрощенное описание, как все на самом деле работает. Основные отличия заключаются в следующем:

- Сообщение `cmpctblock` может также включать в себя некоторые полные транзакции — например, монетарную транзакцию блока.
- Поддержка компактных блоков может работать в двух разных режимах:
  - в режиме большой пропускной способности сообщения `cmpctblock` отправляются без предварительного запроса с использованием `inv` или `headers`;

- в режиме с малой пропускной способностью блок `cmpctblock` отправляется только по запросу, в ответ на `inv` или `headers`.

✶ Для экономии трафика в сообщениях `cmpctblock` посылаются не полные, а сокращенные идентификаторы транзакций. Но они достаточно длинные, чтобы почти всегда однозначно идентифицировать фактические транзакции.

Это по-настоящему полезное изменение, которое многие посчитают ценным. Вы выпускаете свое программное обеспечение, и люди начинают его использовать. Конечно, не все обновятся до этой версии. Но если хотя бы один из ваших соседей по сети начнет использовать его, вы выиграете, установив его у себя, потому что требования к пропускной способности между вами и этим соседним узлом уменьшатся. По мере того как все больше и больше узлов начнет использовать компактные блоки, ваши требования к общей пропускной способности будут снижаться.

Вы не внесли никаких изменений в правила согласования. Блоки проверяются с использованием вашего программного обеспечения точно так же, как и раньше. Старые узлы будут принимать новые блоки и наоборот.

Хард-форк

Как было сказано раньше, хард-форк — это изменение ПО, которое ослабляет правила согласования. Новые блоки, созданные новыми узлами, могут отклоняться старыми узлами. В примере с вегетарианским рестораном хард-форком станет ситуация, когда в этом ресторане начнут подавать мясо.

Предположим, вы изменили максимально допустимый вес блока — это обсуждалось в разделе «Увеличение максимального размера блока» в главе 10 — с 4 000 000 до 8 000 000 весовых единиц. Это позволяет добавлять больше транзакций в каждый блок. С другой стороны, более высокое ограничение может отрицательно повлиять на некоторые узлы в сети Биткоин, как говорилось в главе 10.

Как бы то ни было, вы вносите это изменение и начинаете использовать его в сети Биткоин. Когда ваш узел получает блок от старого узла, вы принимаете его, потому что блок определенно имеет вес $\leq$8 000 000 весовых единиц; старый узел не будет создавать или пересылать блоки с весом больше 4 000 000 единиц.

Предположим, что вы майнер, использующий новый Биткоин. Вам повезло найти действительное доказательство работы, и вы публикуете свой блок. Вес этого блока определенно будет ≤8 000 000 весовых единиц, но может быть больше 4 000 000 единиц. Если блок соответствует условию ≤4 000 000 весовых единиц, он будет принят старыми узлами. В противном случае старые узлы отклонят его. Ваш блокчейн будет отличаться от блокчейна на узлах, использующих старый Биткоин. Вы вызвали расщепление блокчейна (рис. 11.5).

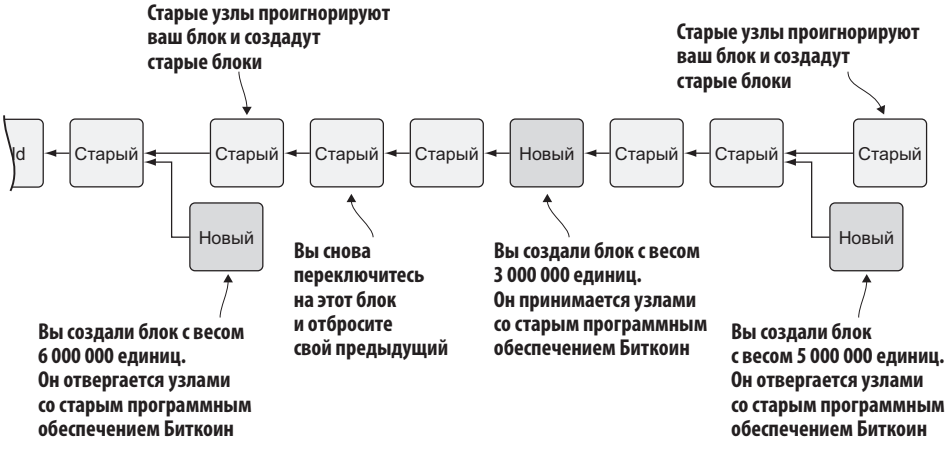


Рис. 11.5. Узел, использующий новый Биткоин, проиграл узлам, использующим старый Биткоин. Узлы со старым программным обеспечением отвергнут ваш блок, нарушающий правило, которое требует, чтобы вес блока был ≤ 4 000 000 весовых единиц

Когда ваш новый узел найдет новый блок, он может быть отклонен старыми узлами, в зависимости от выполнения условия ≤4 000 000 единиц. Вы впустую потратите много электроэнергии и времени на добычу блоков, которые были отклонены, потому что они не попадут в основную цепочку.

Но предположим, что майнерам, обладающим большей долей хешрейта, понравилась ваша версия программы Биткоин, и они начали использовать ее вместо старой версии. Что произойдет в этом случае? Давайте посмотрим (рис. 11.6).

Когда новый узел добудет большой блок, все новые узлы будут пытаться следовать за цепочкой с этим блоком, а все старые узлы будут продолжать цепочку, следующую за последним блоком, который считается действительным, согласно старым правилам.

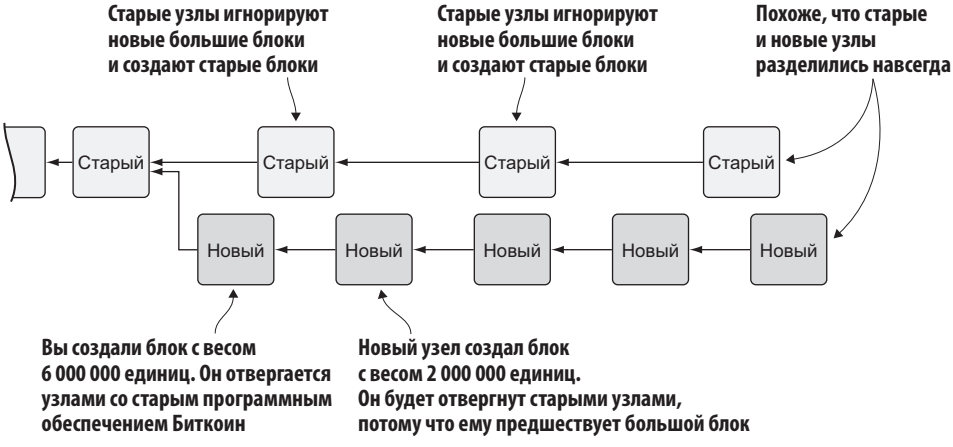


Рис. 11.6. Узлы с большей долей хешрейта используют новую версию Биткоин. Похоже, что это вызовет неустранимое расщепление цепочки

Со временем новые узлы выиграют, потому что обладают большей долей хешрейта, и создадут больше блоков, чем старые узлы. Ветвь, созданная новыми узлами, останется нетронутой, потому что будет обладать бóльшим накопленным доказательством работы.

Новые узлы создадут продолжительное расщепление цепочки. Но если некоторые майнеры решат вернуться к старой версии Биткоин или в гонку вступят новые майнеры, использующие старые узлы, в результате чего старая версия снова получит большую долю хешрейта, цепочка новых блоков может столкнуться с проблемами, как показано на рис. 11.7.

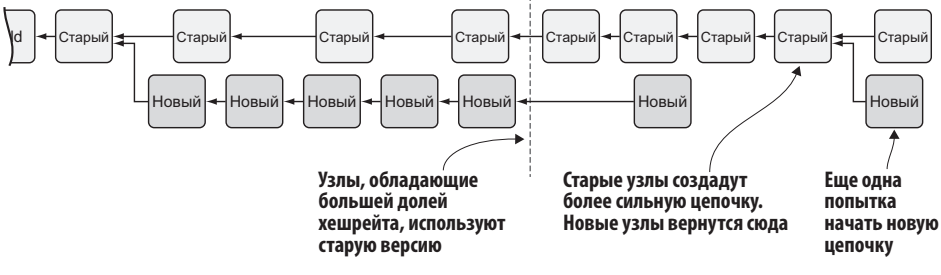


Рис. 11.7. Новая цепочка будет стерта, потому что цепочка, созданная старой версией Биткоин, окажется сильнее

Если старые узлы получат большую долю хешрейта, они все равно догонят новые узлы и превзойдут их. Новые узлы подтвердят этот факт, переклю-

чившись обратно на майнинг по старой цепочке. Мы говорим, что ветвь, созданная новыми узлами, стирается в момент реорганизации блокчейна.

Защита от стирания

Блоки, созданные старыми узлами в только что описанном сценарии хард-форка, всегда совместимы с новыми узлами. Это означает, что есть риск реорганизации нового блокчейна.

Это относится не ко всем хард-форкам. Предположим, вы решили изменить хеш-функцию доказательства работы, заменив двойной SHA256 одинарным. После этого новые блоки всегда будут отклоняться старыми узлами, и наоборот, старые блоки всегда будут отклоняться новыми узлами. Такое изменение гарантированно предотвратит реорганизацию старой ветки. Оно по своей природе защищает от стирания, но многие изменения таковыми не являются.

Примером изменения, которое по своей природе не защищает от стирания, может служить альтернативная криптовалюта под названием *Bitcoin Cash*. Она была создана в результате хард-форка Bitcoin Core в блоке на высоте 478559 1 августа 2017 года. Главными изменениями, которые внесла Bitcoin Cash, были увеличение максимального размера базового блока и удаление поддержки segwit из кода. Это сделало старую цепочку совместимой с новыми узлами и уязвимой для стирания. Чтобы обезопасить новую версию Биткоин от стирания в ходе реорганизации, Bitcoin Cash добавила *защиту от стирания*, потребовав, чтобы первый блок после расщепления имел размер больше 1 000 000 байт (1 Мбайт). Смотрите рис. 11.8.

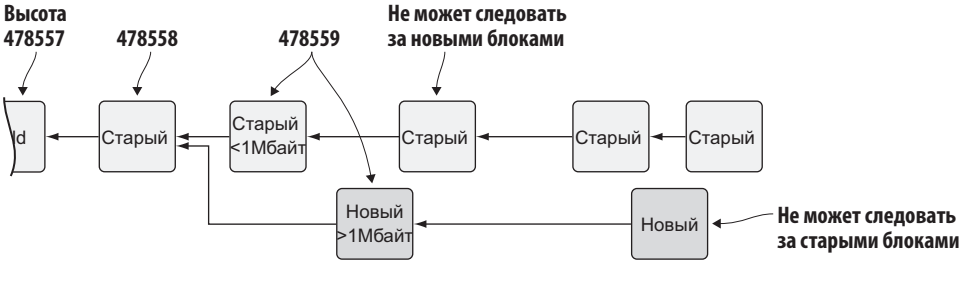


Рис. 11.8. Bitcoin Cash защищает от стирания, потребовав, чтобы первый блок после расщепления имел размер больше 1 Мбайт

В результате новые узлы *не могут* вернуться к старой ветке Биткоин, потому что она имеет блок размером меньше 1 Мбайт на высоте 478559.

Софт-форки

Мы уже несколько раз обсуждали софт-форки в этой книге. Софт-форк — это такое изменение в правилах согласования, после внедрения которого новые блоки будут приниматься старыми узлами. Правила согласования ужесточаются. В примере с вегетарианским рестораном софт-форком была бы ситуация, когда ресторан меняет вегетарианское меню на веганское.

Segwit — пример софт-форка. Это изменение тщательно прорабатывалось, чтобы старые узлы не терпели неудачу при проверке блоков, содержащих segwit-транзакции. Все старые узлы будут принимать любые действительные новые блоки и включать их в блокчейн.

С другой стороны, старый узел *может* создать блок, недопустимый с точки зрения новых правил Биткоин. Например, майнер, не поддерживающий segwit-транзакции, может включить в свой блок транзакцию, которая тратит segwit-выход, как если бы это был выход «любой может потратить» (рис. 11.9).

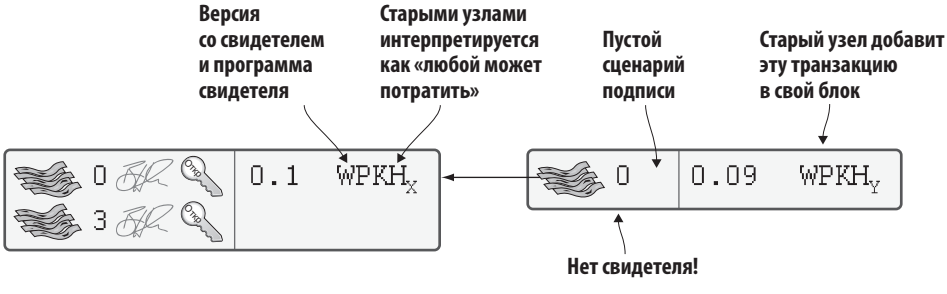


Рис. 11.9. Старый майнер рассматривает segwit-выход как выход «любой может потратить» и добавляет в блок транзакцию, которая расходует его именно таким способом

Предположим, что есть только один майнер с небольшим хешрейтом, использующий новую версию Биткоин. Также предположим, что старые майнеры создали блок, недопустимый с точки зрения новых правил, как в предыдущем примере с не-segwit-транзакцией. Старые узлы создали блок, недействительный для нового майнера. Новый майнер отклонит недействительный старый блок. Это та точка, где блокчейн разделится на две части (рис. 11.10).

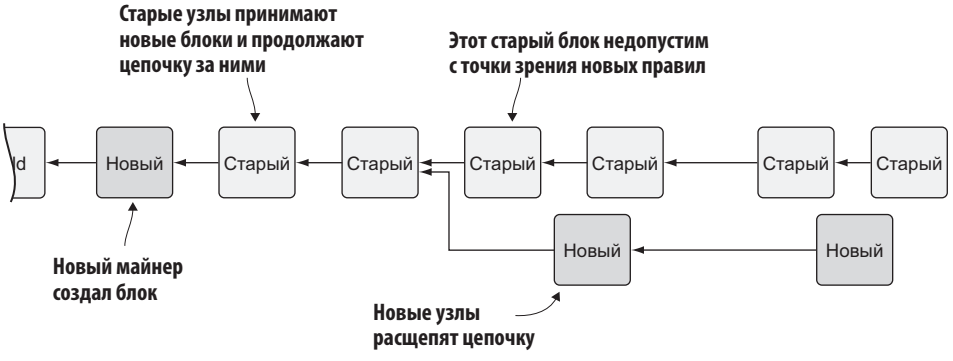


Рис. 11.10. Софт-форк может вызвать расщепление цепочки, если старые узлы произведут блок, недопустимый с точки зрения новых майнеров

В этой ситуации старая цепочка может быть стерта в результате реорганизации. Предположим, что все больше майнеров принимают решение о переходе на новую версию Биткоин, в результате чего новые правила поддерживаются все большей долей хешрейта. Через некоторое время мы, вероятно, увидим реорганизацию (рис. 11.11).

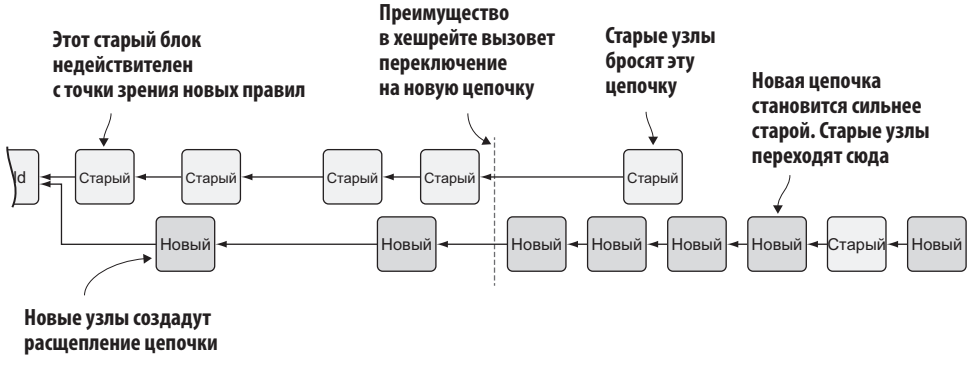


Рис. 11.11. По мере распространения новой версии Биткоин ветвление вызовет реорганизацию старых узлов

Новая ветвь станет более сильной, поэтому оставшиеся старые майнеры покинут свою ветвь и начнут работать с той же ветвью, что и новые узлы. Но как только старый узел создаст блок, недопустимый для новых узлов, он потеряет вознаграждение за блок, потому что тот не будет принят в новую ветвь.

Отличия между софт- и хард-форком

Давайте еще раз посмотрим, что отличает софт-форк от хард-форка. Как правило:

- Хард-форк *ослабляет* правила. Увеличение максимального веса блока — это хард-форк.
- Софт-форк *ужесточает* правила. Внедрение segwit — это софт-форк.

Это простой, но верный признак. А теперь перечислим эффекты из-за расщепления цепочки, вызванного форками:

- *Хард-форк* — новая ветвь может быть стерта во время реорганизации. Чтобы избежать этого, используется защита от стирания. Старая ветвь не может быть стерта.
- *Софт-форк* — старая ветвь может быть стерта во время реорганизации. Старую ветвь невозможно защитить от стирания, потому что иначе ветвление станет жестким. Напомню, что, согласно определению, в случае софт-форка старые узлы считают новые блоки допустимыми.

Повторение транзакции

КОЛЕБАНИЯ КУРСА

Расщепление цепочки может серьезно повлиять на курс биткоинов в старой ветви. Стоимость одной монеты в новой ветви может быть неизвестна и зависит от того, насколько широко продаются эти монеты.

Независимо от причин, вызвавших расщепление, эффект будет тем же самым. Пользователи получат у себя две версии UTXO: одна с доступными для расходования выходами из старой цепочки, а другая — из новой. Фактически образуются две криптовалюты, bitcoin old и bitcoin new — старый и новый биткоин (рис. 11.12).

Предположим, что произошло расщепление блокчейна, как показано на рис. 11.12, а вы хотите заплатить за книгу в интернет-магазине, используя старый биткоин (bitcoin old), потому что именно так захотели в книжном магазине.

Вы создаете свою транзакцию и отправляете ее. Старые узлы в сети примут вашу транзакцию, потому что она расходует UTXO, существующий на этих узлах. Но транзакция *также будет действительной на новых узлах*, потому что эти узлы имеют точно такие же наборы UTXO (рис. 11.13).

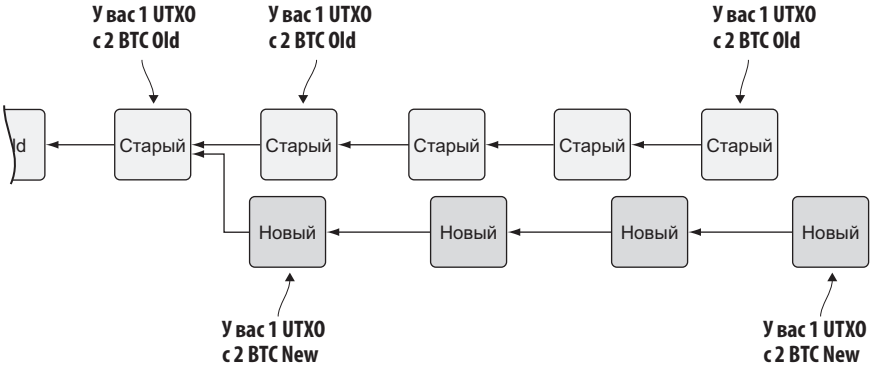


Рис. 11.12. После расщепления цепочки фактически образуются две версии UTXO

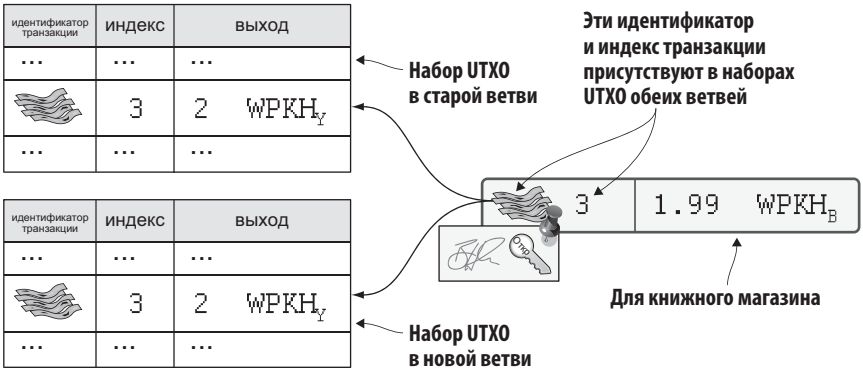


Рис. 11.13. Транзакция с платежом книжному магазину, действительная в обеих ветвях, старой и новой

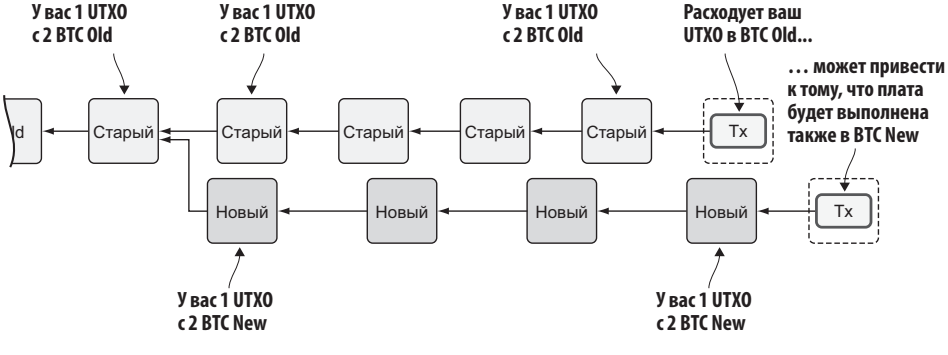


Рис. 11.14. Повторение транзакции вынуждает вас выполнить оплату в обеих валютах

Если ваша транзакция попадет как к новому, так и к старому майнеру, она, скорее всего, будет помещена в обе ветви блокчейна. А это для вас нежелательно. Ваша транзакция была *повторена* в ветви Bitcoin New (рис. 11.14).

Защита от повторения

Для защиты пользователей от повторения транзакций во время расщепления блокчейна из-за хард-форка можно изменить формат транзакций в новой цепочке так, чтобы транзакция была действительной только в одной ветви.

Когда происходило отделение криптовалюты Bitcoin Cash в отдельную цепочку, разработчики позаботились о том, чтобы старые транзакции считались недействительными на новых узлах, а новые транзакции — недействительными на старых (рис. 11.15).

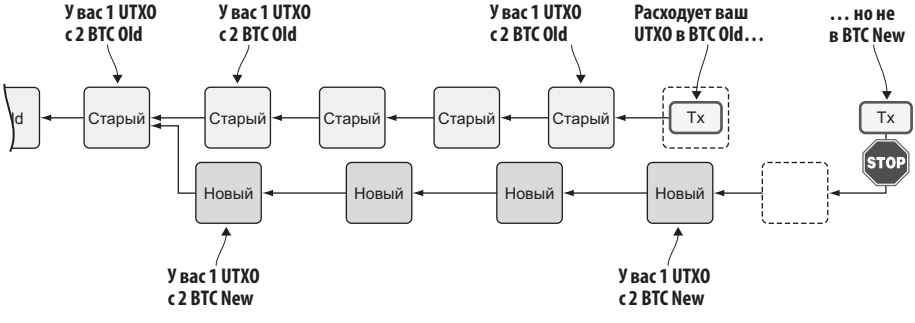


Рис. 11.15. С защитой от повторения транзакция считается действительной только в одной из ветвей

Для этого в подписях транзакций в новой ветви должен использоваться новый `SIGHASH`-тип — `FORKID`. Этот тип не делает ничего, но из-за него транзакция будет считаться недействительной в старой и действительной в новой цепочке. Если транзакция не использует тип `FORKID`, она будет считаться действительной в старой и недействительной в новой цепочке.

Конечно, использование нового `SIGHASH`-типа для подписей — не единственный способ защититься от повторения. Тот же эффект могут дать любые изменения, делающие транзакции действительными только в одной цепочке. Например, можно потребовать, чтобы новые транзакции вычитали `1` из идентификатора входной транзакции. Предположим, что UTXO содержит идентификатор транзакции, выход которой вы хотите потратить:

```
6bde18fff1a6d465de1e88b3e84edfe8db7daa1b1f7b8443965f389d8decac08
```

Тогда, чтобы потратить UTXO в старой цепочке, вы используете этот хеш во входе вашей транзакции. А чтобы потратить UTXO в новой цепочке, вы используете

```
6bde18fff1a6d465de1e88b3e84edfe8db7daa1b1f7b8443965f389d8decac07
```

Имейте в виду, что это всего лишь первый пришедший на ум пример, а не полноценное предложение.

Механизмы обновления

До сих пор все несрочные обновления в Биткоин производились с использованием софт-форка. Обеспечить безопасность софт-форка — сложная задача, что способствовало развитию используемых для этого механизмов.

Основное беспокойство при выполнении софт-форка состоит в том, что блокчейн может расщепиться на две части и надолго остаться в таком состоянии. В этом случае фактически будут существовать две криптовалюты.

Это может привести к путанице: биржи должны будут решить, какую ветвь считать истинной и какую поддерживать в обслуживании. Пользователи должны знать, что произошло расщепление, чтобы они могли избежать отправки денег не в ту ветвь. Торговцы должны убедиться, что взимают валюту или валюты, которые предполагали получить. Расщепление цепочки блоков может привести также к значительному изменению курса криптовалюты.

Использование сигнальной монеты — BIP16

Когда в 2012 году вводились в действие платежи p2sh, сообщество Биткоин не имело опыта в обновлении. Нужно было найти способ избежать расщепления блокчейна. Сообщество реализовало *способ уведомления* о софт-форке с использованием сигнальной монеты. Новые майнеры сообщали о поддержке p2sh, поместив строку `/P2SH/` в монетарные транзакции произведенных ими блоков (рис. 11.16).

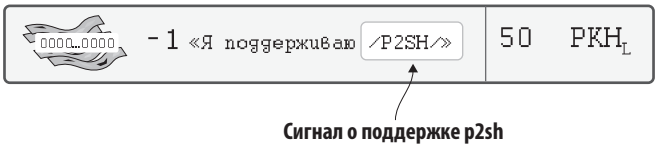


Рис. 11.16. Майнер сигнализирует о поддержке p2sh, поместив строку /P2SH/ в сценарий подписи монетарной транзакции

В определенный день разработчики Биткоин проверили, содержат ли хотя бы 550 блоков из последних 1000 строку `/P2SH/`. Это условие было выполнено, поэтому разработчики выпустили новую версию программного обеспечения, которое с 1 апреля 2012 года, в день флага, начало применять правила p2sh.

Все прошло без сучка и задоринки; майнеры быстро приняли софт-форк, и вся сеть обновилась в разумные сроки. Расщепления не произошло, потому что больше 50% хешрейта было обновлено до внесения в систему изменений.

СОФТ-ФОРКИ, АКТИВИРУЕМЫЕ ПОЛЬЗОВАТЕЛЯМИ

Метод развертывания, когда не только майнеры, но и пользователи начинают применять правила, стал называться *софт-форком, активируемым пользователями*. Мы поговорим об этом позже в этой главе.

Использование увеличенного номера версии блока — BIP34, 66 и 65

Раньше я лишь вскользь упоминал о том, что заголовок блока включает номер версии (рис. 11.17). Номер версии кодируется в первых 4 байтах перед хешем предыдущего блока.

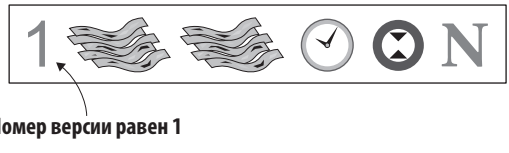


Рис. 11.17. Заголовок содержит номер версии блока. Первый блок имеет версию 1

Версия — единственное, что отсутствовало в предыдущих примерах заголовков блоков. Вот как в действительности устроен 80-байтовый заголовок блока в Биткоин:

```
4 байта с номером версии
32 байта с идентификатором предыдущего блока
32 байта с корнем дерева Меркла
4 байта с отметкой времени
4 байта с целью
4 байта с временным номером

Всего 80 байт
```

Номер версии блока тоже можно использовать для уведомления о поддержке каких-то новых возможностей.

Первый софт-форк с использованием номера версии блока для уведомления было сделано в 2013 году. Это ветвление добавило правило, согласно которому монетарная транзакция во всех новых блоках должна включать высоту блока (рис. 11.18).

BIP34

Это предложение по улучшению, «Block v2, Height in Coinbase», описывает, как сохранить высоту блока в монетарной транзакции и как внедрить это изменение с использованием номера версии.

Активация софт-форка была выполнена в несколько шагов, а чтобы избежать расщепления цепочки, для уведомления использовался номер версии блока.

1. Новые майнеры увеличили номер версии блока с 1 до 2 (рис. 11.19). Обратите внимание, что это происходило постепенно, по мере того как все больше узлов переходило на новую версию программного обеспечения Биткоин.

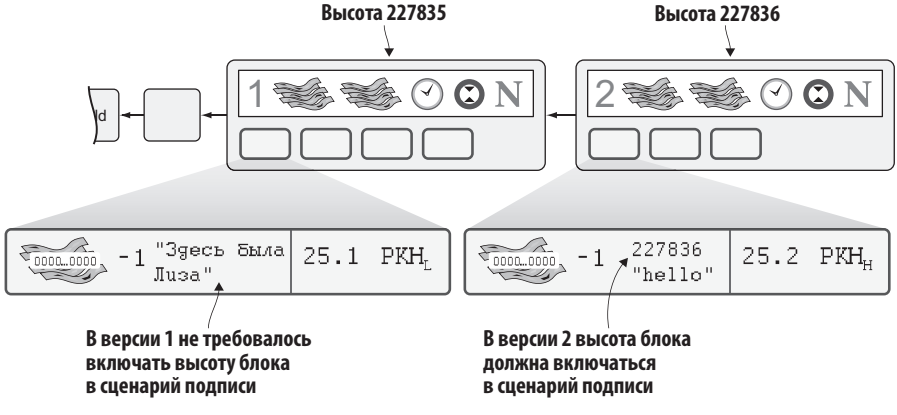


Рис. 11.18. Предложение BIP34 потребовало сохранять высоту блока в монетарной транзакции

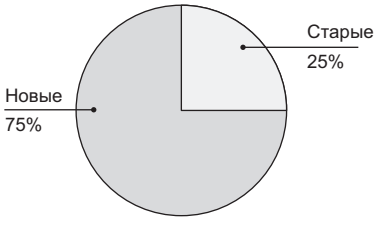
Рис. 11.19. Майнеры, выполнившие софт-форк, сигнализируют о поддержке новых возможностей изменением номера версии блока

2. Ожидание, пока число блоков с номером версии 2 в последней 1000 не превысило 750. После достижения этого порога можно было с уверенностью предполагать, что хешрейт обновившихся майнеров примерно равен 75%.

3. Начало отклонения вновь созданных блоков с номером версии 2, которые не содержат высоты блока в монетарной транзакции. Эти блоки ложно уведомляют о поддержке BIP34.

4. Ожидание, пока число блоков с номером версии ≥ 2 в последней 1000 не превысило 950. Когда это случилось, можно было с уверенностью предполагать, что хешрейт обновившихся майнеров примерно равен 95%.

5. Начало отклонения всех новых блоков с номером версии 1. Все блоки с номером версии, произведенные майнерами после начала этого этапа, отбрасывались, потому что 95% хешрейта начали отвергать эти блоки. Остальные майнеры, не обновившие свое программное обеспечение к этому моменту, очень быстро сделали это, чтобы не тратить впустую средства на майнинг бесполезных блоков.



На шаге 1 ничего не изменилось. Продолжают действовать старые правила Биткоин. Но когда в последней 1000 блоков появилось 750 блоков с номером версии 2, был сделан следующий шаг. С этого момента все узлы, участвующие в софт-форке, начали включать в монетарную транзакцию каждого блока с номером версии 2 его высоту. Если встречался блок, не соответствующий этому условию, он отбрасывался. Причина в том, что узлы могут случайно или преднамеренно использовать номер версии 2 для других целей, не связанных с софт-форком. Правило 75% удаляет ложные срабатывания перед оценкой правила 95%.

С этого момента некоторые майнеры, использующие старое программное обеспечение, могут вызвать расщепление цепочки, создав блок с номером версии 2, который нарушает правило «наличие высоты в монетарной транзакции» (рис. 11.20).

Старые майнеры продолжат строить цепочку за этим блоком, тогда как новые майнеры будут строить цепочку за предыдущим блоком. Новые майнеры, даже с учетом «ложных» сигналов о поддержке версии 2, *наверняка* будут иметь больший суммарный хешрейт, построят более сильную цепочку и сотрут ветвь, создаваемую старыми майнерами.

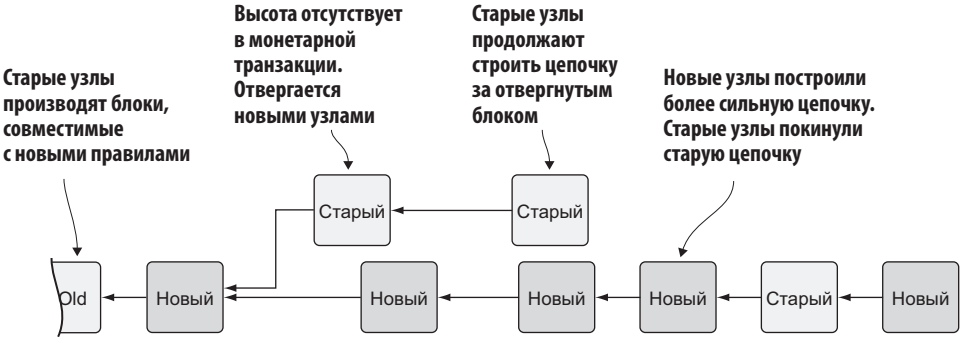
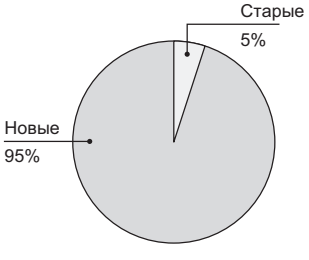


Рис. 11.20. Старые узлы могут вызвать расщепление цепочки, но это расщепление едва ли будет длиться долго



Когда большая часть блоков — 95% из последних 1000 — будет содержать сигнал в виде номера версии 2, начнется следующий и последний шаг, шаг 5. С этого момента все блоки с номером версии < 2 будут отбрасываться.

Зачем нужны все эти этапы? Не совсем понятно, зачем понадобилось правило 75%, которое, как было описано, удаляет ложные сигналы. Внедрение изменения вполне можно было бы провести с единственным правилом 95%. Мы не будем изучать обоснование правила 75%, а просто примем, что оно использовалось в развертывании этого и некоторых других изменений. В табл. 11.2 перечислены софт-форки, которые были введены с использованием этого механизма.

Таблица 11.2. Изменения, внедрявшиеся с использованием увеличения номера версии блока

BIP	Название	Дата	Номер версии блока
BIP34	«Block v2, Height in Coinbase»	март 2013	2
BIP66	«Strict DER Encoding»	июль 2015	3
BIP65	`OP_CHECKLOCKTIMEVERIFY`	декабрь 2015	4

Только что описанный механизм обновления называется *активированием майнерами*. Майнеры начинают применять новые правила, и все или большинство полных узлов следуют за ними, потому что новые блоки принимаются как старыми, так и новыми полными узлами.

Использование отдельных битов в номере версии блока — BIP9

Разработчики Биткоин накопили богатый опыт в ходе предыдущих софт-форков. Им предстояло решить несколько проблем:

- в каждый период времени развертываться могло только один софт-форк;
- номера версий блоков нельзя повторно использовать для других целей.

BIP9

Это предложение по улучшению определяет, как использовать поле в заголовке с номером версии блока для одновременного развертывания нескольких софт-форков.

Самая досадная проблема — невозможность одновременного развертывания нескольких софт-форков. Это связано с тем, что предыдущие механизмы развертывания, такие как описанный в BIP34, проверяли условие: больше или равен номер версии определенному числу, например 2.

Предположим, требуется развернуть оба предложения — BIP34 и BIP66 — одновременно. Для развертывания BIP34 будет использоваться версия блока 2, а для развертывания BIP66 — версия 3. Мы не сможем избирательно сигнализировать о поддержке только BIP66. Присваивая блоку номер версии 3, мы также сообщим о поддержке BIP34, потому что номер версии 3 больше или равен 2.

Разработчики выдвинули предложение по улучшению Биткоин BIP9, в котором описали процесс одновременного развертывания нескольких софт-форков.

Этот процесс тоже использует номер версии блока, но по-другому. Разработчики решили изменить способ интерпретации байтов в номере версии. Версии блоков, в которых старшие 3 бита имеют значение `001`, обрабатываются особым образом.

Во-первых, все такие версии блоков больше 4, потому что наименьший номер версии с тремя старшими битами `001` равен `20000000`, что намного больше, чем `00000004`. Поэтому блоки с поддержкой BIP9 всегда будут поддерживать уже развернутые BIP34, 66 и 65. Отлично.

29 битов, расположенных правее крайних левых битов `001`, можно использовать для одновременной сигнализации о поддержке до 29 софт-форков. Каждый из этих 29 битов в номере версии можно использовать для независимого развертывания одного или группы изменений (рис. 11.21). Если бит

установлен в 1, значит, майнер, создавший блок, поддерживает изменение, представляемое этим битом.

Рис. 11.21. Номер версии блока интерпретируется по-разному. Каждый из 29 битов справа может сигнализировать о поддержке разных предложений

При развертывании каждого изменения должны быть определены следующие параметры:

- *название* — короткое и описательное название изменения;
- *бит* — номер бита, сигнализирующего о поддержке изменения;
- *время начала* — время, с которого начнется мониторинг поддержки изменения майнерами;
- *тайм-аут* — время, по истечении которого попытка развернуть изменение будет признана неудачной.

Развертывание выполняется в несколько этапов (рис. 11.22). Состояние обновляется *после каждого периода корректировки сложности*.

- `DEFINED` — начальное состояние. Означает, что с момента начала корректировка еще не выполнялась.
- `STARTED` — ожидание, когда не менее 1916 (95%) блоков в последнем периоде корректировки будут содержать признак поддержки.
- `LOCKED_IN` — льготный период, чтобы дать остальным майнерам, не сообщившим о поддержке изменения, время на обновление ПО. Если они не уложатся в это время, их блоки будут отклоняться.
- `ACTIVE` — новые правила вступили в силу.

СРАВНЕНИЕ ЗНАЧЕНИЙ ВРЕМЕНИ

Для сравнения значений времени для определения времени начала и тайм-аута всегда используется медианное время в прошлом, как описано в разделе «Правила для отметок времени» в главе 7.

- `FAILED` — превышено время до наступления состояния `LOCKED_IN`. Если условия выполнятся одновременно, например, подтвердится правило 95% и наступит тайм-аут, предпочтение будет отдано тайм-ауту как более приоритетному событию.

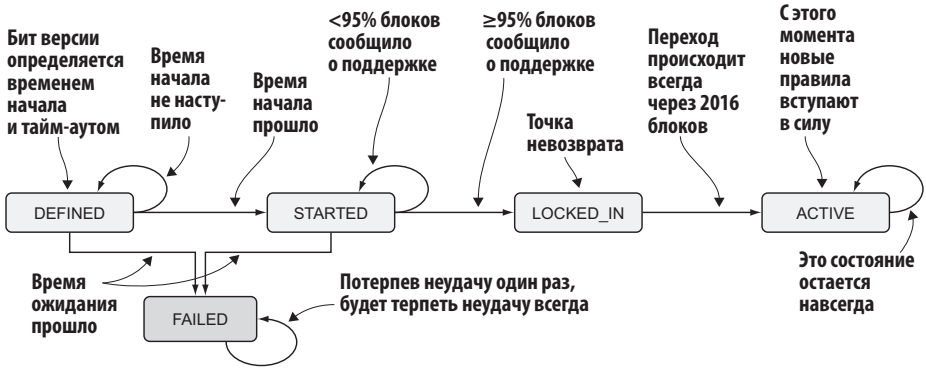


Рис. 11.22. Переход между состояниями происходит через каждые 2016 блоков

Когда развертывание достигло состояния `ACTIVE` или `FAILED`, бит, используемый для сигнализации о поддержке, должен быть сброшен в 0, чтобы затем его можно было использовать повторно для развертывания других изменений.

Использование BIP9 для развертывания относительной временной блокировки

Рассмотрим порядок развертывания изменений с использованием битов в номере версии на примере развертывания относительной временной блокировки. Разработчики этой новой особенности определили следующие параметры BIP9:

```
название:     csv
бит:          0
время начала: 2016-05-01 00:00:00
тайм-аут:      2017-05-01 00:00:00
```

BIP68, 112 И 113

В действительности эта «особенность» является группой предложений BIP, определяющих порядок работы относительной временной блокировки.

Тайм-аут был установлен равным одному году с момента начала, что дало майнерам около года, чтобы закончить переход.

На рис. 11.23 показано, как происходили переходы между состояниями.

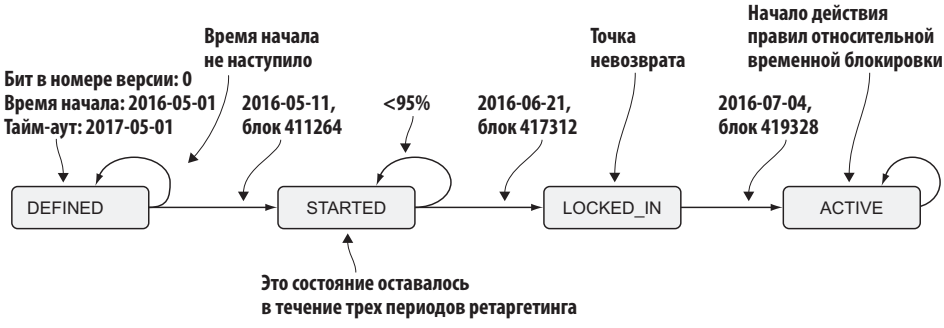


Рис. 11.23. Успешное развертывание csv в соответствии с BIP9

Развертывание этого изменения прошло быстро и гладко. Чтобы набрать 95% майнеров, перешедших на новое программное обеспечение, потребовалось всего три периода корректировки.

К сожалению, не все изменения развертываются так же гладко.

Использование BIP9 для развертывания segwit

Поддержка segwit, описанная в главе 10, также внедрялась с использованием BIP9. Но ее развертывание прошло не так, как ожидалось. Развертывание этого изменения начиналось так же, как развертывание `csv`. Для этого развертывания были выбраны следующие параметры:

```
название:     segwit
бит:          1
время начала: 2016-11-15 00:00:00
тайм-аут:     2017-11-15 00:00:00
```

Была выпущена новая версия Bitcoin Core с этими параметрами развертывания segwit. Пользователи перешли на новую версию довольно быстро, но майнеры почему-то проявили нерешительность. Уровень поддержки застрял на отметке около 30%, и процесс развертывания надолго замер в состоянии `STARTED`, как показано на рис. 11.24.

При развертывании segwit возник риск сбоя — переход в состояние `FAILED` по истечении тайм-аута. Если бы это произошло, пришлось бы начать новый цикл развертывания, который мог бы занять еще один год.

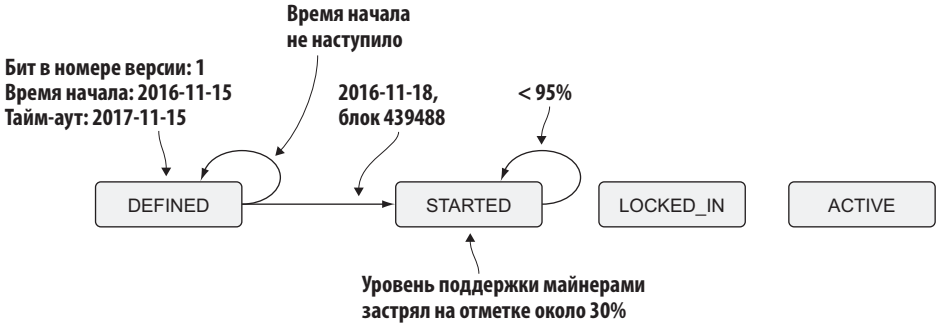


Рис. 11.24. Развертывание поддержки segwit протекало не так, как ожидалось

Конфликт интересов

Одновременно с предложением о внедрении segwit обсуждалось другое предложение, известное как *Segwit2x*. В нем предлагалось сначала активировать segwit, *а уже затем* увеличить максимальный вес блока с использованием хард-форка в дополнение к увеличению максимального размера блока, которое являлось частью segwit. Это предложение предполагало использовать BIP9 с битом 4 в номере версии. Команда Bitcoin Core не проявила интереса к этому предложению, но репозиторий программного обеспечения Bitcoin Core был скопирован с именем btc1 группой людей, которые использовали его для реализации предложения. Порог должен был составить 80% от последних 2016 блоков для перехода на поддержку segwit. Это предложение получило широкую популярность среди майнеров.

Казалось, желания полных узлов и майнеров не совпадают. Ходили слухи и теории о том, что стало причиной такого расхождения. Но мы не будем опираться на слухи, нас интересуют только факты.

Софт-форк, активируемый пользователями

Посреди всего этого появилось еще одно предложение, BIP148, начать отбрасывать блоки без сигнала в бите 1 (segwit) с 1 августа 2017 года. В результате узлы с поддержкой BIP148 будут на 100% готовы принять BIP141, что

приведет к окончательному переходу на поддержку BIP141 всего за два периода корректировки. Это то, что известно как *софт-форк, активируемое пользователями*. Пользователи — те, кто владеет полными узлами, — приняли коллективное решение, что начнут применять новые правила, и если майнеры не будут соблюдать их, блоки будут отбрасываться. Мы еще вернемся к софт-форкам, активируемым пользователями, в конце этой главы.

BIP148 стал попыткой принудительного развертывания segwit, несмотря на нежелание майнеров.

Некоторые группы, особенно команда Bitcoin Core, считали это предложение слишком рискованным, потому что его воплощение может вызвать расщепление цепочки, если большая часть хешрейта не поддержит переход на использование segwit. Но была также группа людей, желающих несмотря ни на что внедрить BIP148. Это вызвало некоторое беспокойство в сообществе Биткоин.

Предложение объединить группы

Развертывание segwit практически застопорилось, потому что появилось альтернативное предложение segwit2x, которое назрело, как казалось многим майнерам, и имелась группа нетерпеливых пользователей, жаждущих внедрить segwit с помощью BIP148.

Чтобы избежать тайм-аута при развертывании segwit, что еще больше задержало бы внедрение этой особенности, и возможного расщепления блокчейна по BIP148, а также чтобы угодить желающим внедрить предложение segwit2x, было написано новое предложение — BIP91, удовлетворившее все группы. Его внедрение предполагалось провести с использованием BIP9 и нестандартным порогом:

```
название:     segsignal
бит:          4
время начала: 2017-06-01 00:00:00
тайм-аут:     2017-11-15 00:00:00
период:       336 блоков
порог:        269 блоков (80%)
Прекращает действовать, когда segwit (bit 1) перейдет в состояние LOCKED_IN или
FAILED.
```

Это предложение действовало немного иначе, чем обычное развертывание с помощью BIP9. В нем использовался более короткий период — 336 блоков вместо 2016 и более низкий порог — 80% вместо 95%.

В период активности это предложение действовало как BIP148. Все блоки без установленного бита 1 (segwit) отклонялись. Обратите внимание, что такое поведение было совместимо с BIP148 и segwit2x. О его поддержке сигнализировал бит 4 — тот же, что сигнализировал о поддержке segwit2x, и принудительно включал действие правил segwit в отношении блоков без установленного бита 1.

Предложение BIP91 было реализовано не в Bitcoin Core, а в скопированной версии Bitcoin Core. Эта версия быстро получила широкое распространение среди майнеров, и 21 июля 2017 года развертывание BIP91 перешло в состояние `LOCKED_IN` (рис. 11.25).

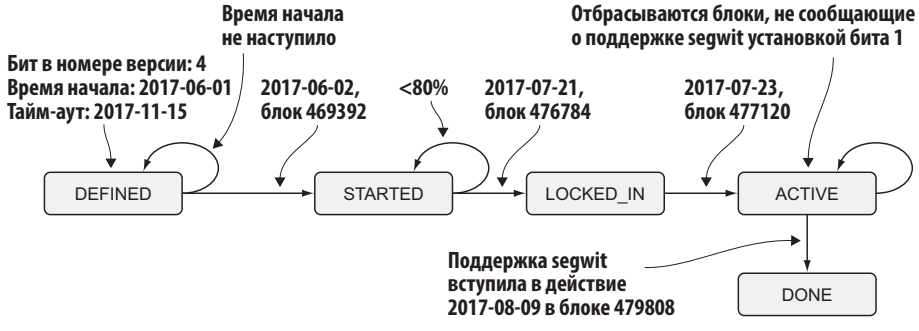


Рис. 11.25. Состояние развертывания BIP91 обновляется через каждые 336, а не через обычные 2016 блоков. Оно прошло очень быстро

Активация произошла через три дня после перехода в состояние `LOCKED_IN`. Обратите внимание: предложение BIP91 принимали в основном майнеры. Рядовые пользователи обычно используют программное обеспечение Bitcoin Core, не поддерживающее BIP91.

После активации BIP91 майнеры начали отбрасывать блоки без установленного бита 1, сигнализирующего о поддержке развертывания `segwit`. В результате блоки со сброшенным битом 1 не попали в самую сильную цепочку, что быстро заставило остальных майнеров обновить свое программное обеспечение, чтобы избежать напрасного расходования ресурсов на майнинг недопустимых блоков.

Майнеры массово начали сигнализировать о поддержке segwit, устанавливая бит 1, и в результате 9 августа 2017 года развертывание предложения BIP141 перешло в состояние `LOCKED_IN` и в состояние `ACTIVE` — 24 августа 2017 года, как показано на рис. 11.26.

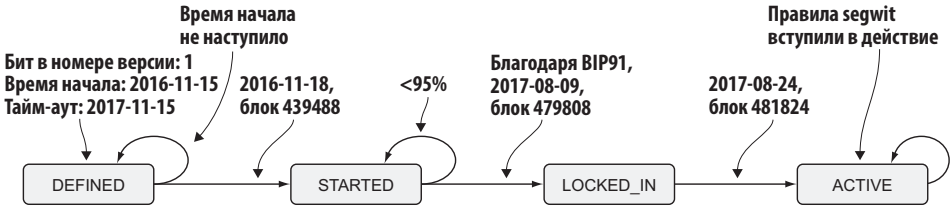


Рис. 11.26. Наконец, благодаря BIP91 предложение segwit было активировано

Обычным пользователям, не занимающимся майнингом, продавцам и биржам не нужно было ничего предпринимать, чтобы остаться в самой сильной цепочке, потому что их программное обеспечение (обычное ПО с поддержкой segwit) следовало за самой сильной, действительной цепочкой. Это означало, что развертывание предложения BIP141 перешло в состояние `LOCKED_IN` и затем `ACTIVE` для всех пользователей и майнеров одновременно.

Извлеченные уроки

События, происходившие во время развертывания segwit, имели неожиданные последствия. Мало кто думал, что майнеры откажутся принять BIP141. Но это случилось.

Стало ясно, что BIP9 не является идеальным способом развертывания софт-форка. Он позволяет заблокировать софт-форк всего 5% хешрейта. Любой майнер, контролирующий более 5% общего хешрейта, сможет заблокировать обновление системы.

Как отмечалось в разделе «Доверие Лизе» в главе 5, мы платим майнерам за правильное и честное подтверждение транзакций. Мы платим им не за определение правил, а за их соблюдение. Все правила устанавливаются коллективно, вами и мной, посредством выбираемого нами программного обеспечения Биткоин.

Подумайте над этим.

Софт-форки, активируемые пользователями

Чтобы подчеркнуть важность экономического большинства (вы, я и все остальные, использующие Биткоин) и лишить майнеров возможности блокировать предложения, за которые выступает это большинство, люди стали больше задумываться о софт-форках, активируемых пользователями.

Давайте рассмотрим вымышленный пример софт-форка, активируемого пользователями.

Предположим, что 99% пользователей Биткоин (конечные пользователи, биржи, продавцы и т. д.) решили изменить правило — например, уменьшить размер блоков. Эти изменения являются софт-форком. Также предположим, что никому из майнеров не нужно уменьшение блоков, поэтому они все отказались подчиняться. Предположим также, что 99% полных узлов, из числа не занимающихся майнингом, обновили свое ПО, чтобы отклонять все большие блоки после достижения определенной высоты.

Что произойдет, когда эта высота будет достигнута? Майнеры, производящие большие блоки, создадут блокчейн, который пользователи сочтут недействительным (рис. 11.27).

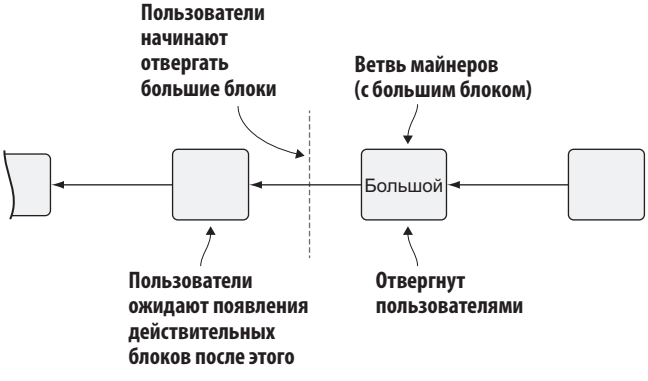


Рис. 11.27. Пользователи начали отвергать большие блоки. Они не видят новых действительных блоков, но видят множество недействительных (слишком больших) блоков

Сумма вознаграждения за блок в цепочке майнеров будет неизвестна, потому что биржи не работают с этой цепочкой. Майнеры не смогут потратить свое вознаграждение за блоки, чтобы оплатить счета за электроэнергию. Даже если поставщик электроэнергии принимает биткоины, майнеры не смогут оплатить электричество вознаграждением за блоки, потому что поставщик электроэнергии отвергнет блоки майнера как недействительные. Поставщик электроэнергии тоже является пользователем Биткоин, помните?

Но если найдется единственный майнер, который решит последовать требованиям пользователей, он окажется единственным производителем блоков, которые примут пользователи (рис. 11.28).

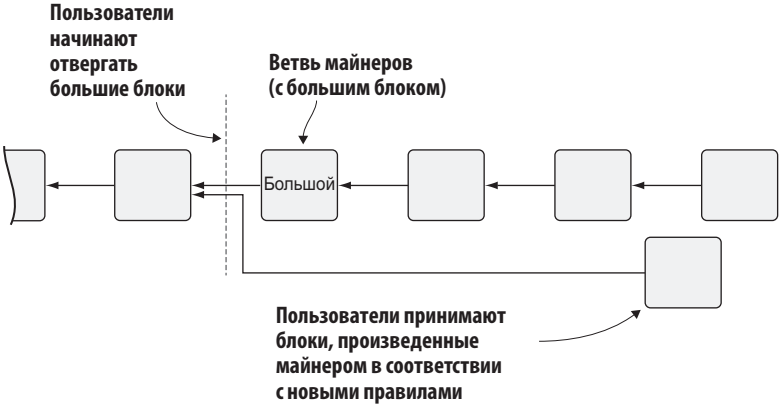


Рис. 11.28. Один майнер решил последовать требованиям пользователей и оказался единственным производителем маленьких блоков. Этот майнер сможет оплатить свои счета

Этот единственный майнер получит вознаграждение за созданный блок, потому что экономически активное большинство примет его. Блоки в цепочке майнеров (большие блоки) останутся бесполезными, потому что никто их не принимает. Кроме того, один майнер, производящий маленькие блоки, сможет взимать больше комиссионных, чем раньше, потому что общий объем пространства блоков уменьшился, оттого что максимальный вес блока стал меньше и оттого что общее количество блоков уменьшилось.

Некоторые майнеры, производившие большие блоки, заметят, что у них быстро заканчиваются деньги, и решат переключиться на ветвь, принимаемую пользователями (рис. 11.29).

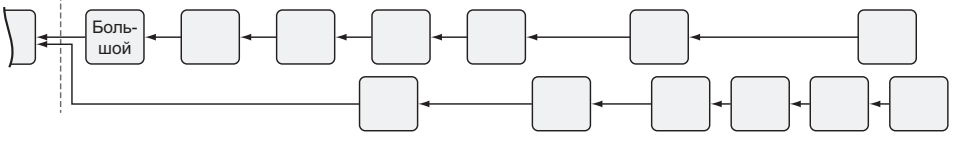


Рис. 11.29. Некоторые майнеры заметили, что работать с ветвью пользователей выгоднее

С переходом майнеров в ветвь пользователей она станет сильнее, чем ветвь с большими блоками. Когда это произойдет, ветвь с большими блоками будет стерта (рис. 11.30), а оставшиеся майнеры автоматически переключатся на ветвь с малыми блоками.

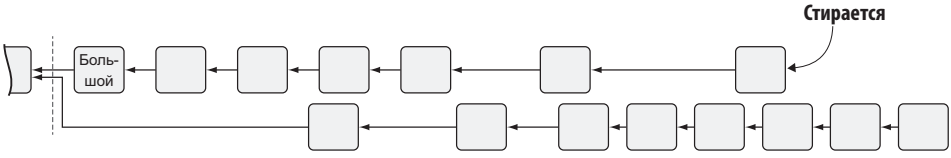


Рис. 11.30. Ветвь пользователей станет сильнее, и ветвь с большими блоками будет стерта

Пользователи победили.

Один из первых софт-форков в Биткоин — развертывание BIP16 (p2sh) — был софт-форком, активируемым пользователями. Развертывание выполнялось вручную, в том смысле, что разработчики в определенный день вручную подсчитывали количество блоков, сигнализировавших о поддержке, а затем определили день флага, который они добавили в следующую версию программного обеспечения Биткоин. После этой даты все блоки, не соответствующие новым правилам, стали отклоняться узлами, использующими это программное обеспечение.

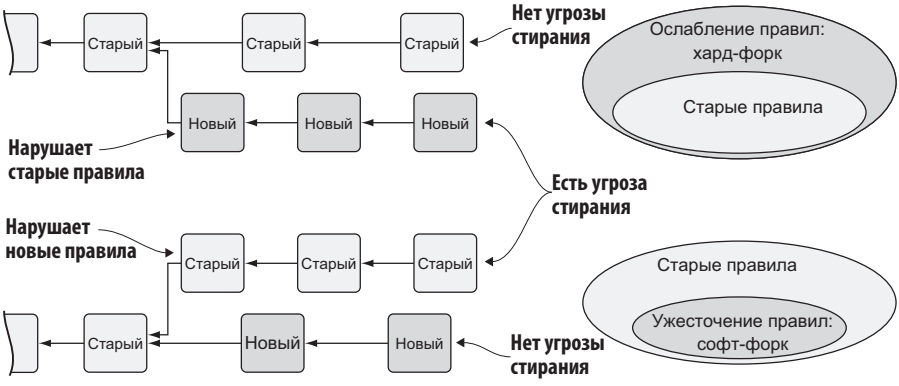
Чтобы учесть уроки, извлеченные из недавнего развертывания segwit, начал создаваться новый механизм развертывания, работа над которым еще продолжалась, когда писались эти строки. Он широко известен под названием «софт-форк, активируемое пользователями». Идея в том, чтобы начать с развертывания в стиле BIP9, но если развертывание не перейдет в состояние `LOCKED_IN` до наступления тайм-аута, блоки, не сигнализирующие о поддержке ветвления, начинают отбрасываться. Это обеспечит 100%-ную поддержку, потому что несовместимые блоки перестанут учитываться и развертывание перейдет в состояние `LOCKED_IN`.

Повторение

В этой главе рассказывалось о различных форках, а также о том, как выполнить софт-форк без расщепления блокчейна. Мы поговорили о нескольких софт-форках, активируемых майнерами, и о нескольких софт-форках, активируемых пользователями.

Хард- и софт-форки можно проиллюстрировать следующим рисунком (с. 457).

В *хард-форк* правила *смягчаются*, из-за чего новый блок может оказаться недействительным с точки зрения старых правил. В случае расщепления блокчейна новая ветвь может быть стерта старой ветвью.

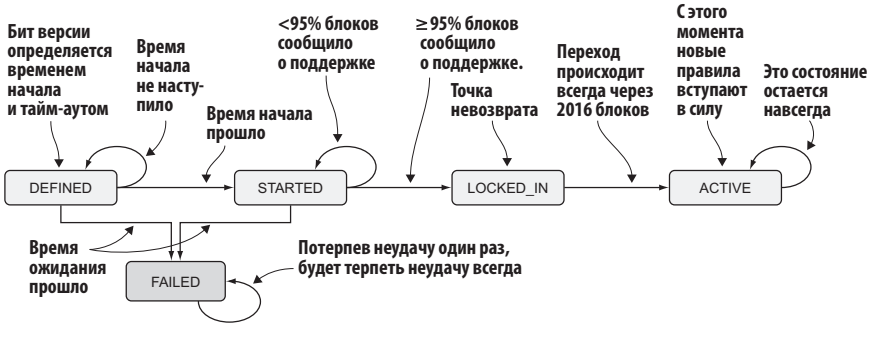


В *софт-форке* правила *ужесточаются*. Старые блоки могут оказаться недействительными с точки зрения новых правил. В случае расщепления блокчейна старая ветвь может быть стерта.

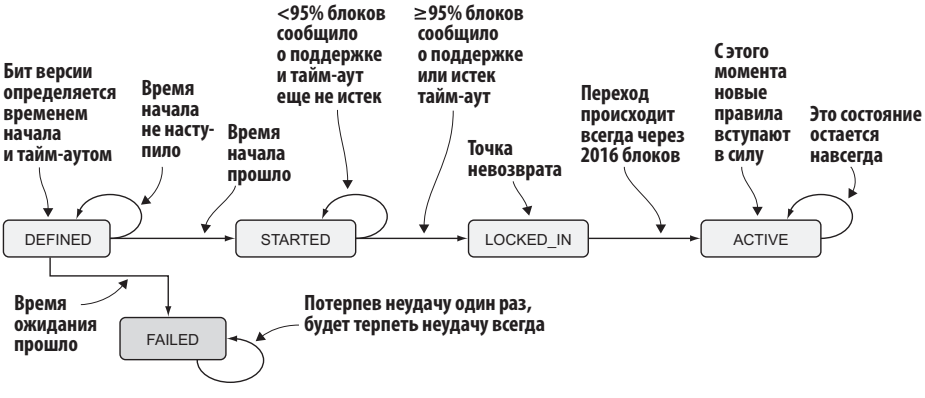
В жестком ветвлении новую ветвь можно защитить от стирания, сознательно сделав ее несовместимой со старой ветвью. Например, Bitcoin Cash требует, чтобы первый блок после расщепления имел базовый размер ≥ 1 000 000 байт, что недопустимо с точки зрения старых правил. В софт-форке нет возможности защитить старую ветвь от стирания.

При проведении софт-форка необходимо соблюдать осторожность, чтобы не расщепить блокчейн. Если произойдет расщепление и обе ветви останутся активными в течение значительного периода времени, это вызовет массу неприятностей у пользователей, бирж, майнеров и т. д.

В *софт-форке, активируемом майнерами*, майнеры сигнализируют о своей поддержке; когда, например, 95% блоков будет содержать сигнал о поддержке, после льготного периода начнут применяться новые правила. Этот процесс стандартизован предложением BIP9.



В *софт-форке, активируемом пользователями*, пользователи начинают применять правила в определенный день (или с определенной высоты блока). На момент написания этих строк разрабатывался стандарт развертывания изменений, который, вероятно, совместит в себе BIP9 и софт-форк, активируемое пользователями.



Отличие от развертывания на основе стандарта BIP9 состоит в том, что процесс софт-форка, активируемый пользователями, гарантированно перейдет в состояние `ACTIVE`, после того как узел войдет в состояние `STARTED`. В состоянии `STARTED` у майнеров есть шанс довести развертывание до состояния `LOCKED_IN`; но если этого не произошло и истекло время ожидания, то полные узлы (включая майнеров), поддерживающие обновление, в любом случае начнут применять правила.

Для развертывания BIP16 — p2sh — использовалось софт-форк, активируемое пользователями, но оно было проведено вручную. Кроме того, сообщество не имеет реального опыта работы с софт-форками, активируемыми пользователями.

Упражнения

Для разминки

11.1. Софт-форк — это изменение правил согласования, но какие отличительные особенности характеризуют софт-форк?

11.2. Предположим, что хард-форк вызвал расщепление блокчейна и новая ветвь получила 51% хешрейта. Предположим также, что хешрейт новой ветви упал примерно до 45%.

а. Какое событие случится в итоге?

б. Почему я сказал, что событие произойдет *в конечном итоге*? Когда оно фактически произойдет?

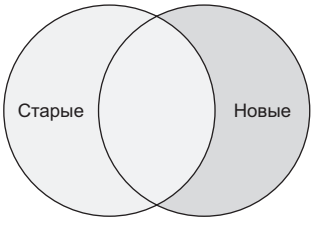
в. Что могут предпринять разработчики новой версии Биткоин, чтобы предотвратить событие?

11.3. Предположим, что старый узел вызвал расщепление блокчейна в ходе софт-форка, в котором 80% хешрейта поддерживают новую версию Биткоин. Как долго будет существовать старая ветвь после расщепления? Поясните свой ответ.

11.4. Предположим, что вы пытаетесь провести софт-форк в соответствии с BIP9. Развертывание только что достигло состояния `LOCKED_IN`. Как долго вы должны ждать перед началом применения ваших правил?

Придется пораскинуть мозгами

11.5. Предположим, что ветвление изменяет правила согласования так, что старые узлы могут создавать блоки, недопустимые для новых узлов, а новые узлы могут создавать блоки, недопустимые для старых узлов.



Какие узлы (новые, старые, и те и другие или ни те ни другие) смогут вызвать расщепление блокчейна при развертывании этого ветвления?

11.6. Почему желательно иметь подавляющее большинство хешрейта, поддерживающего изменения, производимые в ходе софт-форка, прежде чем начинать применять новые правила?

11.7. Предположим, что хард-форк привело к постоянному расщеплению блокчейна и вы собираетесь произвести оплату новыми биткоинами. Почему в этом сценарии желательна защита от повторения транзакций?

11.8. Предположим, вы хотите развернуть софт-форк в соответствии с BIP9 со следующими параметрами:

```
бит:           12
время начала:  2027-01-01 00:00:00
тайм-аут:      2028-01-01 00:00:00
```

Также предположим, что развертывание находится в состоянии `STARTED`, все 2016 блоков в текущем периоде корректировки были добыты и все они сигнализируют о поддержке изменений с использованием бита 12. Последний (2016-й) блок B_1 в текущем периоде корректировки имеет следующие свойства:

```
временная метка T1:             2027-12-31 23:59:59
медианное время в прошлом MTP1: 2027-12-31 23:59:58
```

Достигнет ли когда-нибудь это развертывание состояния `ACTIVE`?

11.9. Предположим, вы решили провести софт-форк, активируемый пользователями. У вас не получается убедить других пользователей установить ваше программное обеспечение. Что произойдет в день флага, если только небольшой процент (<30%) пользователей решат запустить ваше программное обеспечение?

11.10. Предположим, вы решили провести софт-форк, активируемый пользователями. Многим другим пользователям понравилось предлагаемое вами изменение. Допустим, что 80% пользователей установили вашу версию. Почему майнеры (даже те, кому не нравятся ваши изменения) в конечном итоге переключатся на новые правила, установленные этим софт-форком, активируемым пользователями?

11.11. В предыдущем упражнении ваш софт-форк получил поддержку более 80% пользователей. Предположим, что большинство хешрейта решает следовать вашим новым правилам. Что произойдет с узлами, не занимающимися майнингом, которые не примут ваши изменения?

Итоги

- Расщепление блокчейна при развертывании форка — нежелательное явление, потому что приводит к сбоям в работе сети Биткоин.
- Хард-форк — это изменение в правилах согласования, требующее, чтобы каждый майнер обновил свое программное обеспечение. Иначе произойдет расщепление блокчейна.

- Софт-форк — это изменение в правилах согласования, не требующее одновременного обновления всей сети.
- На случай расщепления блокчейна из-за хард-форка нужна защита от стирания, чтобы новая ветвь не подверглась реорганизации старыми узлами.
- На случай расщепления блокчейна нужна защита от повторения, которая позволит выбрать, в какую ветку должны отправляться транзакции.
- Софт-форк, активируемый майнерами — например, использующее BIP9 для развертывания, — позволяет майнерам развернуть непротиворечащие изменения.
- Софт-форк, активируемый пользователями, дает пользователям возможность развернуть свои изменения. Если новым правилам последует большая доля хешрейта, софт-форк пройдет успешно без длительного расщепления.

A Использование bitcoin-cli

Это приложение продолжает раздел «Запуск своего полного узла» в главе 8. Здесь я покажу, как настроить биткоин-кошелек, получать и отправлять биткоины, а также исследовать блокчейн с помощью `bitcoin-cli`, инструмента командной строки Bitcoin Core.

Обратите внимание, что в этом приложении не рассказывается подробно обо всех особенностях `bitcoin-cli`. Рассматривайте его только как отправную точку, где вы получите основы, необходимые, чтобы начать работу. Я предлагаю не останавливаться на этом и продолжить изучение этого инструмента с помощью других источников информации.

Взаимодействие с bitcoind

При запуске `bitcoind` также запускается веб-сервер, который по умолчанию прослушивает TCP-порт 8332. Когда вы используете клиент `bitcoin-cli`, он подключается к веб-серверу, отправляет вашу команду веб-серверу по протоколу HTTP и отображает соответствующие части ответа.

Например, узнать идентификатор первичного блока (блока на высоте 0) можно с помощью следующей команды:

```
$ ./bitcoin-cli getblockhash 0
```

`bitcoin-cli` создаст HTTP-запрос `POST` с телом

```
{"method":"getblockhash","params":[0],"id":1}
```

и отправит его веб-серверу, действующему в `bitcoind`. Свойство `method` в теле запроса — это команда, которую нужно выполнить, а аргумент `0` передается веб-серверу как массив с единственным элементом.

Веб-сервер обрабатывает HTTP-запрос, получает хеш блока из блокчейна и возвращает следующий HTTP-ответ:

```
{"result":"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
↳"error":null,"id":"1"}
```

после чего `bitcoin-cli` выводит значение свойства `result` в окно терминала:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Это тело HTTP-запроса соответствует стандарту JSON-RPC, который описывает, как клиент может вызывать функции в удаленном процессе с использованием формата записи объектов JavaScript (JavaScript Object Notation, JSON).

Использование curl

Поскольку взаимодействие с процессом `bitcoind` происходит через HTTP, для связи с ним можно использовать любую программу, способную отправлять HTTP-запросы `POST`, например утилиту командной строки `curl`. Но при использовании инструментов, отличных от `bitcoin-cli`, вы должны настроить имя пользователя и пароль для аутентификации на веб-сервере.

Остановите узел командой `./bitcoin-cli stop`. Откройте или создайте, если он отсутствует, файл конфигурации Bitcoin Core ~/.bitcoin/bitcoin.conf и добавьте в него следующие строки:

```
rpcuser=<имя пользователя по вашему выбору>
rpcpassword=<пароль по вашему выбору>
```

Изменив и сохранив файл ~/.bitcoin/bitcoin.conf, запустите узел командой `./bitcoind -daemon`, чтобы активировать изменения.

Вот как я вызвал `getblockhash` с помощью `curl` (символ обратного слеша \ означает перенос команды на следующую строку):

```
curl --user kalle --data-binary \
    '{"method":"getblockhash","params":[0],"id":1}' \
    -H 'content-type: text/plain;' http://127.0.0.1:8332/
Enter host password for user 'kalle':
{"result":"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
↳"error":null,"id":1}
```

БОЛЬШЕ ПАРАМЕТРОВ

Bitcoin Core поддерживает большое количество параметров. Выполните команду `./bitcoind --help`, чтобы получить полный список.

Не забудьте поменять имя пользователя `kalle` на то, которое вы настроили в bitcoin.conf.

Эта команда предложит вам ввести пароль. Введите пароль и нажмите Enter. Веб-сервер вернет точно такой же ответ, как при использовании `bitcoin-cli`, но вам придется просмотреть тело ответа, чтобы определить результат — хеш блока 0.

Графический интерфейс пользователя

В состав Bitcoin Core входит графический интерфейс пользователя (GUI). В этом приложении основное внимание уделяется интерфейсу командной строки `bitcoin-cli` для управления процессом `bitcoind` и отправки ему запросов. Но если вы хотите использовать Bitcoin Core в качестве биткоин-кошелька (а не просто как полный узел), вам будет полезно познакомиться с графическим интерфейсом. Версия Bitcoin Core с графическим интерфейсом позволяет выполнять задачи, наиболее типичные для биткоин-кошелька, но полный доступ ко всем возможностям Bitcoin Core можно получить только с помощью `bitcoin-cli`.

Чтобы использовать графический интерфейс Bitcoin Core, остановите текущий узел и запустите версию с графическим интерфейсом, которая называется `bitcoin-qt`:

```
$ ./bitcoin-cli stop
Bitcoin server stopping
$ ./bitcoin-qt &
```

Если не дать процессу `bitcoind` достаточно времени для завершения работы перед запуском `bitcoin-qt`, вы получите сообщение об ошибке. В этом случае щелкните кнопкой OK и попробуйте через несколько секунд снова выполнить команду `./bitcoin-qt &`.

ПОЧЕМУ -QT?

Графический интерфейс Bitcoin Core реализован с использованием библиотеки QT. Отсюда такое имя команды: `bitcoin-qt`.

`bitcoin-qt` использует тот же каталог для данных, `~/.bitcoin/`, что и `bitcoind`, а значит, `bitcoin-qt` будет использовать уже загруженный и проверенный блокчейн и тот же самый кошелек, что и `bitcoind`. Графический интерфейс — единственное отличие между этими двумя программами.

По умолчанию `bitcoin-qt` не запускает веб-сервер для обработки запросов JSON-RPC, как это делает `bitcoind`. Чтобы получить возможность взаи-

модействовать с `bitcoin-qt` с помощью `bitcoin-cli`, запустите `bitcoin-qt` следующей командой:

```
$ ./bitcoin-qt -server &
```

Знакомство с bitcoin-cli

Итак, предположим, что вы уже запустили Bitcoin Core в фоновом режиме как

```
$ ./bitcoind -daemon
```

Первая важная команда, которую нужно запомнить, — это команда `help`. Если выполнить ее без аргументов, она вернет список всех доступных команд:

```
$ ./bitcoin-cli help
```

Вы получите длинный список команд, сгруппированных по темам — например, `Blockchain`, `Mining` и `Wallet`. Назначение некоторых команд очевидно из их имен, но если вы захотите узнать больше о конкретной команде, вызовите команду `help` и передайте ей аргумент с именем интересующей команды. Например:

```
$ ./bitcoin-cli help getblockhash
getblockhash height

Returns hash of block in best-block-chain at height provided.

Arguments:
1. height         (numeric, required) The height index

Result:
"hash"         (string) The block hash

Examples:
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc": [CA]"1.0", "id":"curltest",
↳"method": "getblockhash", "params": [1000] }' -H 'content-type: text/plain;'
↳http://127.0.0.1:8332/
```

Вызвать `bitcoin-cli` можно двумя способами:

- *С помощью позиционных аргументов*. Смысл аргументов зависит от их относительного местоположения, например `./bitcoin-cli getblockhash 1000`. Это самый распространенный вариант использования `bitcoin-cli`.

- *С помощью именованных аргументов.* Аргументы передаются в командную строку с их именами, например `./bitcoin-cli -named getblockhash height=1000`. Этот способ может пригодиться, когда команда принимает необязательные аргументы и нужно передать, например, второй необязательный аргумент, опустив первый. Примеры такого использования будут показаны далее.

Начало работы

Давайте создадим зашифрованный кошелек и его резервную копию. Затем получим немного биткоинов и отправим эти деньги по другому адресу, попутно разобрав детали транзакций, и все это с помощью `bitcoin-cli`.

Создание зашифрованного кошелька

Когда запускается `bitcoind` (или `bitcoin-qt`), автоматически создается кошелек и сохраняется в файле ~/.bitcoin/wallet.dat. Но этот кошелек не зашифрован, то есть его закрытые ключи и начальное число, используемое для получения пар ключей, как описано в главе 4, хранятся на жестком диске в открытом виде. Давайте рассмотрим некоторые данные такого кошелька:

```
$ ./bitcoin-cli getwalletinfo
{
  "walletname": "",
  "walletversion": 169900,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 0,
  "keypoololdest": 1541941001,
  "keypoolsize": 1000,
  "keypoolsize_hd_internal": 1000,
  "paytxfee": 0.00000000,
  "hdseedid": "bb989ad4e23f7bb713eab0a272eaef3d4857f5e3",
  "hdmasterkeyid": "bb989ad4e23f7bb713eab0a272eaef3d4857f5e3",
  "private_keys_enabled": true
}
```

Команда `getwalletinfo` выводит много разной информации об используемом в данный момент кошельке. Этот автоматически созданный кошелек не имеет названия, поэтому в `walletname` указано пустое имя кошелька.

`balance` — это количество подтвержденных биткоинов, имеющихся у вас (включая неподтвержденные исходящие транзакции), а `unconfirmed_balance` — это сумма входящих неподтвержденных платежей. `immature_balance` актуален только для майнеров и обозначает количество вновь

созданных биткоинов, которые нельзя потратить, пока не минует 100 блоков. Вызовите справку для `getwalletinfo` для получения более подробных сведений об информации, выводимой этой командой.

Чтобы получить зашифрованный кошелек, нужно создать новый кошелек командой `encryptwallet`:

```
$ ./bitcoin-cli -stdin encryptwallet
secretpassword<ENTER>
<CTRL-D>
wallet encrypted; Bitcoin server stopping, restart to run with encrypted wallet.
↳The keypool has been flushed and a new HD seed was generated (if you are
  using HD).
↳You need to make a new backup.[1]
```

Эта команда создаст новый зашифрованный кошелек. Ключ `-stdin` заставляет команду читать пароль из стандартного ввода, то есть команда предложит вам ввести пароль в окне терминала. Завершите ввод пароля, нажав Enter и Ctrl-D. Ключ `-stdin` используется, просто чтобы не указывать пароль в самой команде, потому что большинство командных оболочек, таких как bash, хранят историю команд в файле. Ключ `-stdin` гарантирует, что пароль не попадет в такой файл истории.

Важно создать новый зашифрованный кошелек, а не просто зашифровать существующий, потому что старый кошелек на вашем жестком диске мог быть скомпрометирован. Как отмечено в выводе команды, процесс `bitcoind` был остановлен. В настоящее время Bitcoin Core не может переключиться на новый файл кошелька во время работы.

Давайте запустим `bitcoind` и посмотрим на кошелек:

```
$ ./bitcoind -daemon
Bitcoin server starting
$ ./bitcoin-cli getwalletinfo
{
  "walletname": "",
  "walletversion": 169900,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 0,
  "keypoololdest": 1541941063,
  "keypoolsize": 1000,
  "keypoolsize_hd_internal": 1000,
```

[1] Зашифрованный кошелек; сервер Bitcoin остановлен, запустите его снова, чтобы начать работу с новым зашифрованным кошельком. Пул ключей очищен и сгенерировано новое начальное число HD (если вы используете HD). Не забудьте создать резервную копию. — *Примеч. пер.*

```
  "unlocked_until": 0,
  "paytxfee": 0.00000000,
  "hdseedid": "590ec0fa4cec43d9179e5b6f7b2cdefaa35ed282",
  "hdmasterkeyid": "590ec0fa4cec43d9179e5b6f7b2cdefaa35ed282",
  "private_keys_enabled": true
}
```

Старый незашифрованный файл wallet.dat был затерт новым зашифрованным файлом wallet.dat. Однако старое начальное число было сохранено в новом зашифрованном кошельке на тот случай, если в старом кошельке у вас имелись какие-то средства или вы можете получить средства на адрес старого кошелька в будущем. Значение `0` в атрибуте `unlocked_until` означает, что ваши закрытые ключи зашифрованы паролем, указанным при шифровании кошелька. Теперь вам придется расшифровывать свои закрытые ключи, чтобы получить к ним доступ. Ниже, когда мы будем отправлять свои первые биткоины, я покажу, как это сделать.

Резервное копирование кошелька

Вы создали зашифрованный кошелек, но перед его использованием необходимо сделать резервную копию. В главе 4 мы говорили о мнемонических предложениях, определяемых в BIP39, которые упростили резервное копирование начальных чисел иерархически детерминированных (Hierarchical Deterministic, HD) кошельков. Но эта возможность не реализована в Bitcoin Core по нескольким причинам, в основном из-за того что в мнемоническом предложении отсутствует следующая информация:

- Версия формата начального числа.
- *Даты рождения*, когда было создано начальное число. Если дата рождения неизвестна, придется просканировать весь блокчейн, чтобы отыскать свои старые транзакции. Если дата рождения известна, достаточно просканировать блокчейн, начиная с этой даты.
- Пути вывода, или деривации (derivation paths), используемые для восстановления. Этот недостаток в некоторой степени компенсируется использованием стандартных путей деривации, но не все кошельки реализуют стандарт.
- Другие метаданные, такие как метки для адресов.

Чтобы создать резервную копию кошелька Bitcoin Core, нужно скопировать файл wallet.dat. Будьте осторожны, не копируйте файл средствами операционной системы во время работы `bitcoind` или `bitcoin-qt`, так как в этом случае резервная копия может оказаться в несогласованном состоянии из-за того, что процесс `bitcoind` мог записывать в файл данные, пока вы

его копировали. Чтобы гарантировать получение непротиворечивой копии с работающим Bitcoin Core, выполните следующую команду:

```
$ ./bitcoin-cli backupwallet ~/walletbackup.dat
```

Она потребует от `bitcoind` сохранить копию файла кошелька в walletbackup.dat в вашем домашнем каталоге (при желании можно указать другой путь и имя файла). Файл резервной копии будет точной копией исходного файла wallet.dat. Переместите файл walletbackup.dat в безопасное место: запишите на флешку и заприте ее в банковском сейфе или, например, скопируйте на компьютер в квартире брата.

Получение денег

Итак, мы создали зашифрованный кошелек и его резервную копию. Отлично! Теперь положим в него немного биткоинов. Чтобы получать биткоины, нужен биткоин-адрес, поэтому создадим его:

```
$ ./bitcoin-cli -named getnewaddress address_type=bech32
bc1q2r9mql4mkz3z7yfxvef76yxjd637r429620j75
```

Эта команда создаст bech32-адрес p2wpkh. Если вы предпочли бы другой тип адреса, измените `bech32` на `legacy`, чтобы получить адрес `p2pkh`, или на `p2sh-segwit`, чтобы получить адрес p2wpkh, вложенный в адрес p2sh. Вернитесь к разделу «Еще раз о типах платежей» в главе 10, чтобы освежить в памяти информацию о различных типах платежей и адресов.

Теперь отправим биткоины на этот адрес. Будьте внимательны и отправляйте деньги не на адрес, указанный в этой книге (хотя я с радостью приму их в дар), а на свой адрес, который вы создали сами для своего кошелька полного узла.

Теперь возникает вопрос: где взять биткоины для отправки в кошелек. Получить биткоины можно несколькими способами:

- купить на бирже;
- попросить друга, имеющего биткоины, одолжить или продать вам немного;
- заработать;
- добыть майнингом.

Как получить биткоины — решать вам, а я далее буду предполагать, что на адресе, созданном выше, есть биткоины.

Лично я выполнил перевод на свой новый адрес и затем проверил кошелек:

```
$ ./bitcoin-cli getunconfirmedbalance
0.00500000
```

В СЕТИ

Посетите веб-ресурс 20 (см. приложение B), чтобы узнать, как купить биткоины в своей стране.

Как показывает этот пример, появился входящий платеж в размере 5 mBTC (0,005 BTC), ожидающий подтверждения. Теперь нужно подождать, пока платеж не будет включен в блокчейн, а пока углубимся в транзакцию, выполнив команду `listtransactions`. Вот что вывела эта команда у меня:

```
$ ./bitcoin-cli listtransactions
[
  {
    "address": "bc1q2r9mql4mkz3z7yfxvef76yxjd637r429620j75",
    "category": "receive",
    "amount": 0.00500000,
    "label": "",
    "vout": 1,
    "confirmations": 0,
    "trusted": false,
    "txid": "ebfd0d14c2ea74ce408d01d5ea79636b8dee88fe06625f5d4842d2a0ba45c195",
    "walletconflicts": [
    ],
    "time": 1541941483,
    "timereceived": 1541941483,
    "bip125-replaceable": "yes"
  }
]
```

Эта транзакция имеет 0 подтверждений и переводит 0,005 BTC. Также можно заметить, что эта транзакция имеет идентификатор `ebfd0d14...ba45c195`.

Рассмотрим эту транзакцию поближе, выполнив команду `getrawtransaction`:

```
$ ./bitcoin-cli getrawtransaction \
    ebfd0d14c2ea74ce408d01d5ea79636b8dee88fe06625f5d4842d2a0ba45c195 1
{
  "txid": "ebfd0d14c2ea74ce408d01d5ea79636b8dee88fe06625f5d4842d2a0ba45c195",
  "hash": "ebfd0d14c2ea74ce408d01d5ea79636b8dee88fe06625f5d4842d2a0ba45c195",
  "version": 1,
  "size": 223,
  "vsize": 223,
  "weight": 892,
  "locktime": 549655,
  "vin": [
    {
      "txid": "8a4023dbcf57dc7f51d368606055e47636fc625a512d3481352a1eec909ab22f",
      "vout": 0,
      "scriptSig": {
```

```
        "asm": "3045022100cc095e6b7c0d4c42a1741371cfdda4f1b518590f1af
        ↳0915578d3966fee7e34ea02205fc1e976edcf4fe62f16035a5389c661844f7189
        ↳a9eb45adf59e061ac8cc6fd3[ALL]
        ↳030ace35cc192cedfe2a730244945f1699ea2f6b7ee77c65c83a2d7a37440e3dae",
        "hex":
        "483045022100cc095e6b7c0d4c42a1741371cfdda4f1b518590f1af0915578d3966
        ↳fee7e34ea02205fc1e976edcf4fe62f16035a5389c661844f7189a9eb45adf59e061
        ↳ac8cc6fd 30121030ace35cc192cedfe2a730244945f1699ea2f6b7ee77c65c83a2d7
        ↳a37440e3dae"
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": 0.00313955,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 6da68d8f89dced72d4339959c94a4fcc872fa089
↳ OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9146da68d8f89dced72d4339959c94a4fcc872fa08988ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1AznBDM2ZfjYNoRw3DLSR9NL2cwwqDHJY6"
        ]
      }
    },
    {
      "value": 0.00500000,
      "n": 1,
      "scriptPubKey": {
        "asm": "0 50cbb07ebbb0a22f11266653ed10d26ea3e1d545",
        "hex": "001450cbb07ebbb0a22f11266653ed10d26ea3e1d545",
        "reqSigs": 1,
        "type": "witness_v0_keyhash",
        "addresses": [
          "bc1q2r9mql4mkz3z7yfxvef76yxjd637r429620j75"
        ]
      }
    }
  ],
  "hex":
  "01000000012fb29a90ec1e2a3581342d515a62fc3676e455606068d3517fdc57cfdb
  ↳23408a000000006b483045022100cc095e6b7c0d4c42a1741371cfdda4f1b518590f1
  ↳af0915578d3966fee7e34ea02205fc1e976edcf4fe62f16035a5389c661844f7189a9
  ↳eb45adf59e061ac8cc6fd30121030ace35cc192cedfe2a730244945f1699ea2f6b7ee77
  ↳c65c83a2d7a37440e3daefdffffff0263ca040000000000001976a9146da68d8f89dced72
  ↳d4339959c94a4fcc872fa0898820a1070000000000016001450cbb07ebbb0a22f11266653
  ↳ed10d26ea3e1d54517630800"
}
```

Эта команда вывела транзакцию целиком в удобочитаемой (по крайней мере, для разработчика) форме. Начнем сначала и пройдемся по самым важным разделам этой транзакции. `txid` — это идентификатор транзакции.

`hash` — двойной хеш SHA256 транзакции целиком, включая свидетеля. В транзакциях без свидетеля значение `hash` равно значению `txid`.

Размер транзакции `size` составляет 223 байта, так же как и виртуальный размер `vsize` (virtual size), который тоже составляет 223 байта. Атрибут `vsize` вычисляется как количество весовых единиц транзакции (892), деленное на 4, поэтому виртуальный размер транзакции без свидетеля (как эта, потому что она тратит выход транзакции без свидетеля) равен ее фактическому размеру `size`.

Время блокировки этой транзакции установлено равным `549655`. Это высота самой сильной цепочки на момент создания транзакции. По этой причине транзакция не может быть включена в блоки до высоты 549656. Это снижает вероятность атаки, когда майнер намеренно пытается реорганизовать блокчейн и включить транзакцию в уже добытый блок.

Далее следует список входов. Эта транзакция имеет один вход, который тратит выход с индексом `0` (`vout`) транзакции с идентификатором `8a4023db...909ab22f`. Вход расходует выход p2pkh.

Порядковый номер входа равен `4294967293` (`ffffffd` в шестнадцатеричном виде). Это означает, что включена временная блокировка (`≤ fffffffe`) и транзакция является заменяемой (`≤ fffffffd`) в соответствии с BIP125. Обобщенное описание значений порядковых номеров можно увидеть в табл. 9.1.

За списком входов следует список выходов транзакции. Эта транзакция имеет два выхода. Первый платит 0.00313955 BTC на адрес p2pkh, который вы раньше не видели. Это, *вероятно*, выход со сдачей. Второй выход посылает 0,005 BTC на ранее созданный адрес p2wpkh.

Теперь посмотрим, подтвердилась ли транзакция. Сделать это можно, например, с помощью `getbalance`. В моем случае, если команда вернет `0.00500000`, значит, транзакция была подтверждена:

```
$ ./bitcoin-cli getbalance
0.00500000
```

Отлично, получение денег подтверждено! Теперь пойдем дальше.

Отправка денег

Мы получили немного биткоинов и теперь хотим отправить их кому-то еще. Отправку можно выполнить командой `sendtoaddress`. Но для начала нужно сделать следующее:

- узнать адрес для отправки;
- определить сумму для отправки: 0,001 BTC;
- как быстро должна быть подтверждена транзакция: транзакция не срочная (нас вполне устроит, если она подтвердится в течение 20 блоков).

Я отправлю биткоины на адрес `bc1qu456...5t7uulqm`, но вы должны использовать другой адрес для отправки. Если у вас нет другого кошелька, создайте новый адрес в Bitcoin Core для экспериментов. Я исказил свой адрес в команде ниже, чтобы вы по ошибке не отправили на него свои деньги.

```
$ ./bitcoin-cli -named sendtoaddress \
    address="bc1qu456w7a5mawlgXXXXXXu03wp8wc7d65t7uulqm" \
    amount=0.001 conf_target=20 estimate_mode=ECONOMICAL
error code: -13
error message:
Error: Please enter the wallet passphrase with walletpassphrase first.[1]
```

Вот те на! Ошибка. Как нетрудно догадаться по тексту сообщения об ошибке, закрытые ключи в файле wallet.dat оказались зашифрованы. Закрытые ключи нужны Bitcoin Core для подписания транзакции. Чтобы открыть доступ к закрытым ключам, их нужно расшифровать. Это делается с помощью команды `walletpassphrase` с параметром `-stdin`, чтобы предотвратить сохранение пароля интерпретатором командной строки, таким как bash:

```
$ ./bitcoin-cli -stdin walletpassphrase
secretpassword<ENTER>
300<ENTER>
<CTRL-D>
```

Последний аргумент, `300`, это время в секундах, в течение которого кошелек должен оставаться открытым. Спустя 300 секунд кошелек автоматически закроется, если вы забудете сделать это вручную. Теперь повторим команду `endtoaddress`:

```
$ ./bitcoin-cli -named sendtoaddress \
    address="bc1qu456w7a5mawlgXXXXXXu03wp8wc7d65t7uulqm" \
    amount=0.001 conf_target=20 estimate_mode=ECONOMICAL
a13bcb16d8f41851cab8e939c017f1e05cc3e2a3c7735bf72f3dc5ef4a5893a2
```

На этот раз команда вывела идентификатор вновь созданной транзакции. Это означает, что деньги благополучно отправлены. Теперь можно закрыть кошелек командой `walletlock`:

```
$ ./bitcoin-cli walletlock
```

[1] Ошибка: сначала введите пароль к кошельку с помощью walletpassphrase. — *Примеч. пер.*

Кошелек закрыт. Снова выведем информацию о транзакции:

```
$ ./bitcoin-cli listtransactions
[
  {
    "address": "bc1q2r9mql4mkz3z7yfxvef76yxjd637r429620j75",
    "category": "receive",
    "amount": 0.00500000,
    "label": "",
    "vout": 1,
    "confirmations": 1,
    "blockhash": "000000000000000000240eec03ac7499805b0f3df34a7d5005670f3a8fa
                  836ca",
    "blockindex": 311,
    "blocktime": 1541946325,
    "txid": "ebfd0d14c2ea74ce408d01d5ea79636b8dee88fe06625f5d4842d2a0ba45c195",
    "walletconflicts": [
    ],
    "time": 1541941483,
    "timereceived": 1541941483,
    "bip125-replaceable": "no"
  },
  {
    "address": "bc1qu456w7a5mawlg35y00xu03wp8wc7d65t7uulqm",
    "category": "send",
    "amount": -0.00100000,
    "vout": 1,
    "fee": -0.00000141,
    "confirmations": 0,
    "trusted": true,
    "txid": "a13bcb16d8f41851cab8e939c017f1e05cc3e2a3c7735bf72f3dc5ef4a5893a2",
    "walletconflicts": [
    ],
    "time": 1541946631,
    "timereceived": 1541946631,
    "bip125-replaceable": "no",
    "abandoned": false
  }
]
```

Новая транзакция — последняя из двух. Она пока не подтверждена, о чем говорит атрибут `"confirmations": 0`. Размер комиссии составил 141 сатоши. Рассмотрим эту транзакцию подробнее:

```
$ ./bitcoin-cli getrawtransaction \
    a13bcb16d8f41851cab8e939c017f1e05cc3e2a3c7735bf72f3dc5ef4a5893a2 1
{
  "txid": "a13bcb16d8f41851cab8e939c017f1e05cc3e2a3c7735bf72f3dc5ef4a5893a2",
  "hash": "554a3a3e57dcd07185414d981af5fd272515d7f2159cf9ed9808d52b7d852ead",
  "version": 2,
  "size": 222,
  "vsize": 141,
  "weight": 561,
  "locktime": 549665,
  "vin": [
```

```
    {
      "txid": "ebfd0d14c2ea74ce408d01d5ea79636b8dee88fe06625f5d4842d2a0ba4
               5c195",
      "vout": 1,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "txinwitness": [
        "30440220212043afeaf70a97ea0aa09a15749ab94e09c6fad427677610286666a3
        ↳decf0b022076818b2b2dc64b1599fd6b39bb8c249efbf4c546e334bcd7e1874115
        ↳da4dfd0c01",

        "020127d82280a939add393ddbb1b8d08f0371fffbde776874cd69740b59e098866"
      ],
      "sequence": 4294967294
    }
  ],
  "vout": [
    {
      "value": 0.00399859,
      "n": 0,
      "scriptPubKey": {
        "asm": "0 4bf041f271bd94385d6bcac8487adf6c9a862d10",
        "hex": "00144bf041f271bd94385d6bcac8487adf6c9a862d10",
        "reqSigs": 1,
        "type": "witness_v0_keyhash",
        "addresses": [
          "bc1qf0cyrun3hk2rshttetyys7kldjdgvtgs6ymhzz"
        ]
      }
    },
    {
      "value": 0.00100000,
      "n": 1,
      "scriptPubKey": {
        "asm": "0 e569a77bb4df5df446847bcdc7c5c13bb1e6ea8b",
        "hex": "0014e569a77bb4df5df446847bcdc7c5c13bb1e6ea8b",
        "reqSigs": 1,
        "type": "witness_v0_keyhash",
        "addresses": [
          "bc1qu456w7a5mawlg35y00xu03wp8wc7d65t7uulqm"
        ]
      }
    }
  ],
  "hex":
  "02000000000010195c145baa0d242485d5f6206fe88ee8d6b6379ead5018d40
  ↳ce74eac2140dfdeb0100000000feffffff02f319060000000000160014bf041f27
  ↳1bd94385d6bcac8487adf6c9a862d10a0860100000000000160014e569a77bb4
  ↳df5df446847bcdc7c5c13bb1e6ea8b024730440220212043afeaf70a97ea0aa09
  ↳a15749ab94e09c6fad427677610286666a3decf0b022076818b2b2dc64b1599
  ↳fd6b39bb8c249efbf4c546e334bcd7e1874115da4dfd0c0121020127d82280a
  ↳939add393ddbb1b8d08f0371fffbde776874cd69740b59e09886621630800"
}
```

Первое, что нужно отметить: идентификатор `txid` и хеш `hash` этой транзакции отличаются. Это объясняется поддержкой segwit. Как рассказывалось в главе 10, структура свидетеля не включается в идентификатор транзакции, чтобы избавиться от проблемы пластичности транзакций, но хеш `hash` в выходе включает ее. Обратите внимание, что значения `size` и `vsize` также отличаются, что ожидаемо для segwit-транзакций. Комиссионные отчисления составили 141 сатоши, как показывает команда `listtransactions`, и размер `vsize` равен 141 виртуальному байту. То есть ставка вознаграждения была выбрана Bitcoin Core как 1 сатоши за виртуальный байт.

Транзакция имеет один вход, который расходует выход `1` транзакции `ebfd0d14...ba45c195`. Вы уже видели этот выход в разделе, где я переводил 0,005 BTC в свой кошелек Bitcoin Core. Поскольку это выход p2wpkh, сценарий подписи (`scriptSig`) пуст, а `txinwitness` содержит подпись и открытый ключ.

Порядковый номер входа 4294967294 — это `fffffffe` в шестнадцатеричном виде. То есть для транзакции включена временная блокировка, но транзакция не может быть заменена в соответствии с BIP125 (возможность замены за плату).

У меня есть два выхода. Первый — это сдача 0,00399859 обратно на мой собственный адрес. Другой — фактический перевод на сумму 0,001 BTC. Давайте еще раз проверим баланс:

```
./bitcoin-cli getbalance
0.00399859
```

Все верно. Мне не нужно ждать подтверждения, чтобы увидеть новый баланс, потому что `getbalance` всегда учитывает мои собственные исходящие неподтвержденные транзакции. Я потратил свой единственный UTXO (0,005 BTC) и создал новый UTXO 0.00399859:

```
Spent:   0.005
Pay:    -0.001
Fee:    -0.00000141
==================
Change:  0.00399859
```

Все точно.

Я показал несколько команд, которые можно использовать для запуска своего узла Bitcoin Core, но далеко не все. Исследуйте справку `./bitcoin-cli help`, чтобы больше узнать о других командах.

Б Решения упражнений

Глава 2

2.1. 256 бит.

2.2. 32 байта.

2.3. Криптографическая хеш-функция.

2.4. `061a` — это `6*256 + (16 + 10) = 1536 + 26 = 1562` в десятичной форме. В двоичной форме `06` — это `0000 0110`, а `1a` — это `0001 1010`, то есть полное двоичное представление будет иметь вид `0000 0110 0001 1010`.

2.5. Нет. Если бы это было возможно, функция не была бы устойчивой к определению второго исходного образа (second-pre-image-resistant).

2.6. Свойства 2 и 4.

2.7. Устойчивость к определению второго исходного образа. Атакующему потребуется найти набор данных, дающий тот же хеш, что и исходный набор: изображение кота.

2.8. Скорость увеличения денежной массы с течением времени уменьшается, потому что вознаграждение Лизе будет уменьшаться вдвое каждые 4 года. Это означает, что максимально возможное общее количество жетонов (CT) составляет примерно 21 000 000.

2.9. Сотрудники имеют доступ к электронной таблице для чтения. Они могут заглянуть в таблицу и убедиться, что Лиза не приписывает себе лишних жетонов.

2.10. Закрытый ключ создается с помощью какого-либо генератора случайных чисел. Самый простой способ — подбросить монету 256 раз, чтобы сгенерировать 256-битный закрытый ключ. Также можно использовать генератор случайных чисел, встроенный в вашу операционную систему.

2.11. Закрытый ключ.

2.12. Хеширование используется, потому что подпись должна быть небольшой и фиксированного размера. Подпись не должна увеличиваться в размерах с увеличением подписываемого сообщения.

2.13. Чтобы украсть жетоны Джона, Мэллори понадобится его закрытый ключ. Ей также потребуется его имя (Джон), чтобы написать письмо Лизе, но эта информация доступна всем в электронной таблице.

2.14. Фред может с помощью вашего открытого ключа зашифровать свое послание и отправить его вам. Вы сможете расшифровать его с помощью своего закрытого ключа.

2.15. Вы должны подписать сообщение своим закрытым ключом и вложить цифровую подпись вместе с посланием в бутылку. Фред сможет затем убедиться, что подпись действительно сделана с помощью вашего закрытого ключа. Для этого он может расшифровать подпись с использованием вашего открытого ключа и сравнив расшифрованный хеш с фактическим хешем сообщения. Если они совпадут, он может быть уверен, что послание написано именно вами.

Глава 3

3.1. Хеш PKH укорачивается, чтобы: а) уменьшить размеры электронной таблицы и б) сделать адреса жетонов на булочки (и адреса в Биткоин) короче и удобнее для записи человеком.

3.2. Да, такое преобразование существует. Существует специальный алгоритм для декодирования base58check.

3.3. Используется отправителем для преобразования адреса получателя в PKH. Отправитель должен послать PKH получателя в электронном письме Лизе.

3.4. Выполним пошаговое base58-кодирование байтов 0047:

1. Удаляем начальные нулевые байты (00). В данном случае имеется один такой байт. Остается байт 47.
2. Преобразуем в десятичное представление: шестнадцатеричное число 47 — это 4 × 16 + 7 = 71 в десятичном представлении.
3. Разделим 71 на 58: 71 = 1 × 58 + 13. Получаем частное 1 и остаток 13.
4. Разделим частное (число 1) на 58: 1 = 0 × 58 + 1. Получаем частное 0 и остаток 1.
5. Находим остатки 13 и 1. Получаем: E и 2.
6. Добавляем 1, обозначающую один байт 00, удаленный на шаге 1. Получаем: E21.
7. Переворачиваем: 12E. Конец.

3.5. 4-байтная контрольная сумма.

3.6. Он должен создать два отдельных платежа в кафе. Например: первый платеж с 2 CT с адреса `@1` и второй платеж с 8 CT с адреса `@2`. Другой способ: перевести 2 CT с адреса `@1` на адрес `@2`, а затем перевести 10 CT в кафе с адреса `@2`.

3.7. Да, можно. Выполнить base58check-кодирование хеша PKH, чтобы получить адрес.

3.8. Нет, потому что в электронной таблице указаны хеши открытых ключей. Так как криптографические хеш-функции являются односторонними, нет никакой возможности получить открытый ключ из его хеша.

3.9. Они могут посмотреть на переводимые суммы. Платежи на сумму 10 CT, вероятнее всего, соответствуют покупкам выпечки.

3.10. Злоумышленник не может украсть жетоны, потому что ему нужен открытый ключ, чтобы воспользоваться уязвимостью в функции создания открытого ключа. Электронная таблица содержит хеши открытых ключей (PKH); злоумышленник не сможет получить открытый ключ из хеша.

3.11. Злоумышленнику нужен закрытый ключ, чтобы подписать мошенническое письмо Лизе. Даже если он сможет обратить результат RIPEMD160, ему все равно потребуется выполнить атаку получения исходного образа SHA256 и получить закрытый ключ по имеющемуся открытому ключу.

Глава 4

4.1. `bitcoin:155gWNamPrwKwu5D6JZdaLVKvxbpoKsp5S?amount=50`

4.2. Каждому символу соответствует 6 бит энтропии, потому что $2^6 = 64$. Десяти таким символам соответствует 60 бит энтропии, или 60 бросков монеты.

4.3. Вот эти четыре проблемы:

- пароли легко забываются;
- хорошую случайность трудно обеспечить;
- с развитием технологий безопасность уменьшается;
- нужно хранить два элемента: резервную копию и пароль. Это увеличивает вероятность потери резервной копии.

4.4. Начальное случайное число создается с помощью генератора случайных чисел — например, путем подбрасывания монеты или с помощью генератора случайных чисел в вашей операционной системе.

4.5. Расширенный закрытый ключ (xprv) состоит из закрытого ключа и цепного кода.

4.6. Расширенный открытый ключ (xpub) состоит из открытого ключа и цепного кода.

4.7. Расширенный закрытый ключ (xprv) `m/2/1` и требуемый индекс, то есть `7`.

4.8. Нет. Чтобы создать расширенный открытый ключ (xpub) `M/2/1/7'` необходим расширенный закрытый ключ (xprv) `m/2/1`. Сначала нужно создать защищенный xprv `m/2/1/7'` из `m/2/1`, используя процедуру создания защищенного xprv, а затем вычислить xpub `M/2/1/7'` из `m/2/1/7'`.

4.9. Вычислить главный xprv можно с помощью следующей процедуры:

1. Из главного xpub `M` получить xpub `M/4` и запомнить левую половину хеша, L_4.
2. Из `M/4` получить левую половину хеша L_{41} для индекса 1.
3. Вычесть L_{41} из закрытого ключа `m/4/1`, чтобы получить закрытый ключ `m/4`.

4. Вычесть L_4 из закрытого ключа `m/4`, чтобы получить закрытый ключ `m`.

5. `m` вместе с цепным кодом из xpub `M` — это главный xprv.

4.10. Да, вы сможете украсть все деньги с любого адреса, потому что есть возможность вычислить главный xprv.

4.11. Жертва могла бы использовать защищенный `m/4'`. В этом случае вы не смогли бы получить главный xprv. Похитив `m/4'/1` и главный xpub, вы сможете украсть только средства под ключом `m/4'/1`. У вас не получится вычислить xpub `M/4'`.

4.12. Сотрудники могут импортировать xpub счета прямых продаж, после чего получат возможность генерировать любые открытые ключи для этого счета и, таким образом, генерировать столько адресов, сколько потребуется, без необходимости знать какие-либо закрытые ключи.

4.13. Ваш кошелек (и кошелек Аниты) может заранее сгенерировать 10 адресов и проверить электронную таблицу на присутствие этих адресов. Если Аните заплатят по одному из этих адресов — например, по первому из них, — ваш кошелек не будет повторно использовать этот адрес в запросе на оплату, отправляемом клиенту, а возьмет следующий неиспользованный адрес.

Глава 5

5.1. Вы могли бы потратить выходы с 4 CT и 7 CT. В результате были бы созданы новые выходы: с 10 CT на адрес кафе и с 1 CT сдачи на ваш адрес.

5.2. Они используются во входах для ссылки на транзакции, выходы которых расходуются.

5.3. Потому что нельзя потратить только часть выхода транзакции. Вы либо тратите выход, либо нет. Если расходуемый выход содержит больше средств, чем сумма платежа, вы должны вернуть себе сдачу.

5.4. Во входах, в сценариях подписи.

5.5. Чтобы проверяющий знал, каким открытым ключом проверять подпись. Нельзя проверить подпись с помощью хеша открытого ключа (PKH), поэтому нужно явно указать открытый ключ в сценарии подписи.

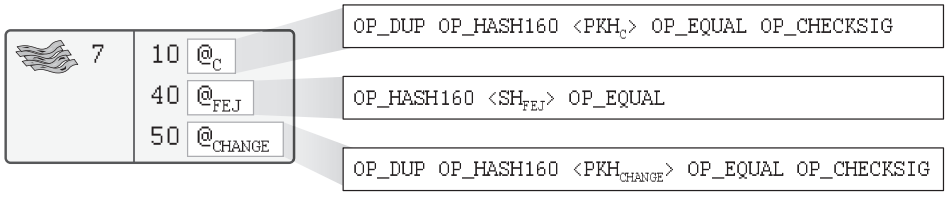
5.6. Сценарии подписи убираются, чтобы проверяющему не требовалось знать, в каком порядке подписывались входы.

5.7. Сценарий открытого ключа имеется в каждом выходе транзакции и содержит вторую часть программы на языке Script. Первая часть будет предоставлена позже, при расходовании выхода.

5.8. Программа на языке Script должна после завершения оставить на вершине стека признак `OK`.

5.9. Адрес p2sh всегда начинается с 3. Также опознать адрес p2sh можно, выполнив base58check-декодирование и посмотрев на первый байт. Если это байт 05, значит, перед вами адрес p2sh.

5.10. Такая транзакция должна иметь один вход и три выхода:



5.11. 10 003 UTXO. Вы удаляете два UTXO, потратив два выхода, и добавляете пять новых UTXO. В результате набор UTXO увеличивается на 3 элемента.

5.12. Сценарий открытого ключа, например, мог бы содержать единственную цифру `1`. Расходующий вход в этом случае мог бы содержать пустой сценарий подписи. Получившаяся полная программа на языке Script просто поместит `1` на стек. Любой результат на стеке, отличающийся от нуля, интерпретируется как признак `OK`.

5.13. `OP_ADD 10 OP_EQUAL`. Этот сценарий сначала сложит два верхних элемента на стеке и поместит результат на стек. Затем сценарий поместит на стек число 10 и сравнит два верхних элемента. Если они равны, на стеке останется признак `OK`.

5.14. Да. Ваш полный узел проверяет все содержимое электронной таблицы, от самой первой транзакции до транзакции, в которой Фаиза перевела вам деньги. Также он проверит (кроме всего прочего), что:

- Лиза создала ожидаемое число монетарных транзакций с правильными суммами;
- сумма средств в выходах в каждой транзакции в электронной таблице не превышает сумму во входах;

- все подписи действительны, начиная от подписи в платеже Фаизы и заканчивая подписями во всей цепочке до всех монетарных транзакций.

5.15. Если имеется несколько UTXO для одного и того же PKH, то после израсходования одного из них безопасность других UTXO для того же PKH ухудшится. Это объясняется удалением слоя защиты, криптографической хеш-функции. С этого момента вы полагаетесь исключительно на безопасность функции создания открытого ключа. Избежать этой проблемы можно, используя уникальные адреса для всех входящих платежей. Тогда все ваши UTXO будут иметь разные PKH.

Глава 6

6.1. Посредством идентификатора предыдущего блока, который является хешем его заголовка.

6.2. В вычислении корня дерева Меркла используются все транзакции из этого блока.

6.3. Подписывая блок, Лиза подтверждает отметку времени, корень дерева Меркла (и косвенно все транзакции в блоке) и идентификатор предыдущего блока (и косвенно всю цепочку перед этим блоком).

6.4. Первая транзакция в каждом блоке — это монетарная транзакция, создающая 50 новых жетонов на булочки и переводящая их на адрес Лизы.

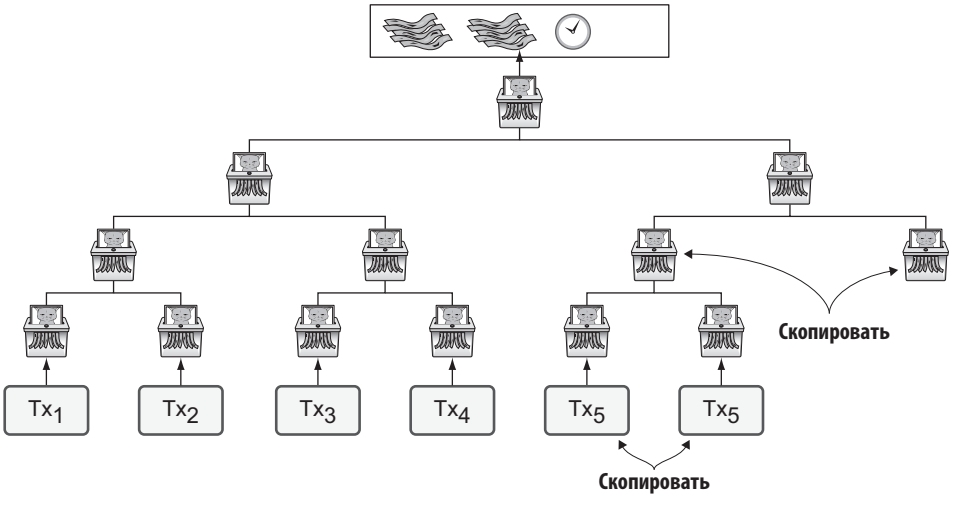
6.5. Все транзакции. Хеш-функции всегда будут давать индексы, содержащие 1, потому что в фильтре Блума нет нулей. Любой элемент в транзакции будет давать положительный результат при проверке.

6.6. Проверяются следующие элементы:

- идентификатор транзакции (txid) вместе с индексом расходуемого выхода;
- все данные в сценариях подписи;
- все данные в сценариях открытого ключа;
- идентификатор транзакции (txid).

6.7. Они не являются стойкими к определению исходного или второго исходного образа. Пространство выходов мало — обычно от нескольких сотен до нескольких тысяч чисел. Требуются доли секунды, чтобы найти исходный образ, например 172.

6.8. Нужно добавить копию самого правого листа, чтобы получилось четное количество листьев. То же касается следующего уровня, где нужно скопировать третий хеш.

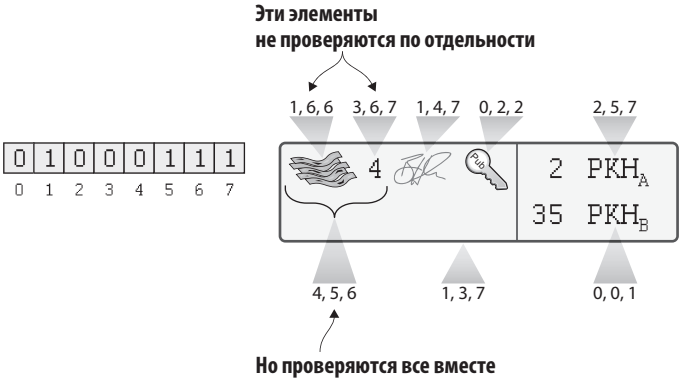


6.9. Украв закрытый ключ Лизы для подписания блоков, вор может создавать блоки от имени Лизы. Кроме того, если злоумышленник заменит открытый ключ Лизы для подписания блоков в одном или нескольких источниках, таких как доска объявлений или внутренняя сеть, злоумышленник сможет обманом заставить людей принять блоки, которые в действительности не подписывались Лизой.

6.10. Лиза может цензурировать сами транзакции, а администратор общей папки может цензурировать блоки.

6.11. а) Да, новый узел, еще не загрузивший все блоки из общей папки, обнаружит наличие двух версий блока. б) Да, старый узел, уже загрузивший оригинальный блок, заметит появление альтернативной версии блока.

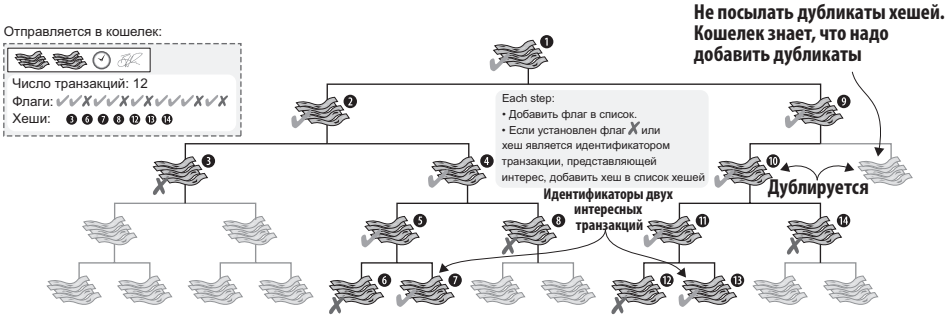
6.12. Биты с индексами `1`, `5`, `6` и `7` установлены в 1, а другие в 0. Полный узел *не* отправит эту транзакцию в легкий кошелек. Ничего из предлагаемого для проверки не хешируется в индексы с битами, равными `1`. Это был вопрос с подвохом, потому что идентификатор расходуемой транзакции и индекс расходуемого выхода не тестируются по отдельности, поэтому 1,6,6 никогда не будет рассматриваться полным узлом.



6.13. Вот это частичное дерево Меркла:

```
Число транзакций: 3
Флаги: ✓✓✗✓✓
Хеши: 3 4 6
```

6.14. Интересными будут сочтены транзакции с номерами 7 и 13 или листья с номерами 6 и 10 слева. Решение уже приводилось в разделе «Обработка тысяч транзакций в блоке», но я повторю его здесь для справки.



6.15. Нужно проверить, что:

- идентификатор транзакции находится в списке хешей;
- корень частичного дерева Меркла совпадает с корнем дерева Меркла в заголовке блока;
- заголовок блока правильно подписан.

Глава 7

7.1. Она единолично решала, какие транзакции подтверждать.

7.2. Возможность цензуры транзакций уменьшается, потому что все майнеры должны войти в сговор, чтобы цензура стала возможной. Иначе все транзакции будут рано или поздно подтверждены кем-то из майнеров, не участвующих в сговоре.

7.3. Майнеры могут лгать, заявляя о выпадении счастливого числа, и вы не сможете доказать, что они лгут.

7.4. Проверить, что идентификатор блока меньше целевого значения в заголовке и что целевое значение — это согласованное целевое значение.

7.5. Многократным изменением одноразового номера и хешированием (двойной SHA256) заголовка блока, пока идентификатор блока (хеш заголовка) не станет меньше целевого значения.

7.6. Ветвь с наибольшим накопленным доказательством работы. Это не обязательно ветвь с большим числом блоков.

7.7. Майнер с хешрейтом 100 Мхеш/с может выполнить в секунду 100 000 000 попыток найти действительное доказательство работы.

7.8. Целевое значение увеличится. Если для производства 2016 блоков потребовалось 15 дней вместо запланированных 14, значит, сложность поиска блоков оказалась слишком высокой и ее следует уменьшить, а это означает увеличение целевого значения.

7.9. 50%. Но если вы планируете сдаться в какой-то момент, ваши шансы уменьшатся.

7.10. Маленький блок быстрее распространяется в сети, потому что для его передачи требуется меньше времени. Маленький блок, вероятно, также быстрее проверяется. Майнеры, вероятно, предпочтут быстрее загрузить и проверить маленький блок и продолжать майнинг в цепочке вслед за маленьким блоком. Все это увеличивает вероятность включения маленького блока в самую сильную цепочку.

7.11. Целевое уменьшится в 3/4 раза. Время создания 2016 блоков составляет 1,5 недели; на создание первых 1008 блоков ушла 1 неделя, а на создание следующих 1008 блоков — 0,5 недели. Поэтому новым целевым значением станет

$$N = O \times \begin{cases} \frac{1}{4} & \text{если } T < 0{,}5 \\ \frac{T}{2} & \text{если } 0{,}5 \le T < 8 = O \times \frac{1.5}{2} = O \times \frac{3}{4}. \\ 4 & \text{если } 8 < T \end{cases}$$

7.12. Селма имеет большую часть хешрейта. Пока она играет по тем же правилам, что и все остальные, она будет хорошо зарабатывать на вознаграждениях за блоки. Когда она нарушит правила, преждевременно изменив целевое значение, все полные узлы, кроме Селмы, начнут отбрасывать ее блоки. Селма продолжит работу над своей ветвью блокчейна со своими новыми правилами, а все остальные будут продолжать свою ветвь, используя старые правила. Ветви будут взаимно несовместимы. Хешрейт старой ветви снизится до 48%, но система от этого не рухнет и все будут продолжать жить, как и прежде. Селма, напротив, будет тратить много электроэнергии и времени на свою новую ветку, и никто не примет у нее ее вознаграждения за блоки. Стоимость добытых ею монет, скорее всего, будет близка к нулю, потому что она не следует правилам. В итоге Селма проиграет.

7.13. Величина платы за байт — метрика, используемая большинством майнеров, — будет очень низкой. Каждый байт транзакции, который майнер помещает в свой блок, приводит к небольшому снижению его конкурентоспособности из-за увеличения блока и, следовательно, замедления его распространения по сети и проверки. Если плата за байт окажется недостаточно высокой, чтобы компенсировать потерю конкурентоспособности, майнер, скорее всего, не включит такую транзакцию в свой блок.

Глава 8

8.1. Общая папка плоха тем, что дает администратору этой общей папки абсолютную власть над тем, какие блоки оставлять в ней. Кроме того, если администратор решит начать майнинг, он сможет устранить всех конкурентов и заполучить всю мощь системы.

8.2. Под пересылкой транзакции или блока подразумевается передача их своим соседям.

8.3. Сообщение `inv` используется, чтобы сообщить соседям, что у вас есть некоторый блок или транзакция; название `inv` происходит от англ. *inventory* (опись, перечень, переучет).

8.4. Он сопоставит транзакцию с фильтром Блума, полученным от кошелька. Если какой-либо проверяемый элемент в транзакции соответствует фильтру, узел пошлет транзакцию в легкий кошелек.

8.5. После сопоставления с фильтром Блума полный узел отправит легкому кошельку сообщение `inv`. Затем кошелек сможет получить транзакцию, если у него ее еще нет.

8.6. Заголовок блока.

8.7. Потому что легкому кошельку кафе не нужно скрывать от своего доверенного узла, какие адреса принадлежат кошельку. Очень большой фильтр Блума используется для экономии мобильного трафика; фильтр Блума, который содержит много нулей, почти не дает ложных срабатываний.

8.8. Проверить подпись программы, используя открытый ключ, который заведомо принадлежит команде разработчиков Bitcoin Core. Это поможет исключить запуск поддельного программного обеспечения.

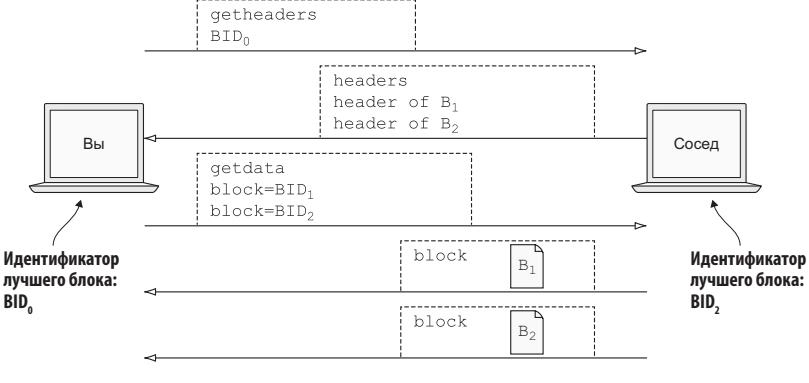
8.9. С помощью сервера DNS можно получить список IP-адресов для начального DNS-имени, настроенного в Bitcoin Core, опросив надежных друзей и использовав жестко закодированные адреса в Bitcoin Core.

8.10. Соседние узлы объявляют о получении новых блоков, посылая сообщения `headers` даже узлам, находящимся на этапе синхронизации.

8.11. Чтобы не допустить получение новых блоков узлом Лизы, потребуется подкупить или запугать кафе, Ци и Тома.

8.12. Она пошлет Рашиду сообщение `inv` с двумя идентификаторами транзакций.

8.13. Ваш узел начнет процедуру синхронизации, изображенную ниже:



Глава 9

9.1. Минимум один из входов должен иметь порядковый номер меньше `ffffffff`.

9.2. Медианная отметка времени по 11 предыдущим блокам должна находиться в прошлом относительно 2019-12-25 00:00:00.

9.3. В самых правых 16 битах порядкового номера.

9.4. По две транзакции в каждой цепочке: одна контрактная транзакция и одна обменная транзакция.

9.5. При хранении данных в виде фиктивных PKH в выходах, данные должны будут вечно храниться в наборе UTXO, поскольку узлы в Биткоин не отличают фиктивные PKH от реальных. Узлы не смогут определить, является ли выход нерасходуемым или нет. В случае с выходом `OP_RETURN` узел знает, что выход является нерасходуемым и, следовательно, его не нужно сохранять в наборе UTXO.

9.6. Если ваша транзакция предлагает слишком маленькую комиссию и застряла в ожидании. В такой ситуации у вас может появиться желание заменить ее новой транзакцией, предлагающей более высокую комиссию.

9.7. Абсолютная временная блокировка: транзакция недействительна до определенной высоты блока или отметки времени. Относительная временная блокировка: вход транзакции недействителен, пока расходуемый выход не будет подтвержден определенным количеством блоков или времени.

9.8. Сценарий погашения содержит две ветви кода. Первая требует наличия двух подписей, вашей и Рут, чтобы потратить 2 BTC. Это можно сделать в любое время. Чтобы потратить 2 BTC с использованием второй ветви, необходимо выполнить все следующие условия:

- дождаться Рождества;
- Бет должна подписать транзакцию;
- вы и Рут должны подписать транзакцию.

Точнее говоря, вы и Рут сможете потратить эти деньги, использовав первую ветвь со следующим сценарием подписи (исключая сценарий погашения):

```
0 <ваша подпись> <подпись Рут> 1
```

Вторая ветвь позволяет потратить только после Рождества:

```
0 <подписи, ваша и Рут > <подпись Бет> 0
```

Самая правя цифра в обоих этих сценариях подписи выбирает ветвь для использования; остальные элементы удовлетворяют требованиям соответствующих ветвей.

Ветвь с временной блокировкой гарантирует, что Бет не сможет вступить в сговор с вами или Рут до Рождества.

9.9. Нет. Сценарий погашения неизвестен узлам до попытки потратить выход. А поскольку невозможно потратить сценарий погашения `OP_RETURN`, узлы никогда не увидят сценарий погашения.

9.10. Получив транзакцию, полный узел хранит ее в памяти, пока она не будет включена в блок. Если поступит вторая конфликтующая транзакция, узел отбросит эту вторую транзакцию и не перешлет ее дальше. Первая транзакция будет считаться «настоящей», а вторая — попыткой повторного расходования. Узлы (включая майнеров) не обязаны следовать этой политике, потому что это просто политика.

9.11. Майнеры могут по своему усмотрению выбирать любые действительные транзакции для включения в свои блоки. Поэтому заменена может быть любая транзакция. Майнер может предложить замену как услугу, то есть выгрузить транзакцию, повторно расходующую выход, с высокой комиссией через веб-сайт майнера, чтобы подтвердить ее в следующем выигравшем блоке майнера.

Конечно, обычным пользователям проще использовать вариант с согласием на замену за плату. Но для мотивированного вора не составит большого труда воспользоваться услугой, как описано выше. Поэтому разница в безопасности не так велика, как вы думаете.

Глава 10

10.1. Сценарии подписи.

10.2. Транзакция T_2, которая тратит выход неподтвержденной транзакции T_1, может стать недействительной, если во время широковещательной передачи T_1 будет заменена на T_{1M} и T_{1M} получит подтверждение. Это вызывает много проблем для контрактов.

10.3. С удвоением числа входов время проверки старых транзакций увеличивается в четыре раза. Это объясняется следующими причинами:

- требуется проверить в два раза больше подписей;
- на проверку каждой подписи требуется в два раза больше времени из-за удвоения размеров транзакции.

10.4. Чтобы убедиться, что транзакция включена в блок, легкий кошелек должен рассчитать идентификатор транзакции, а для этого кошельку нужны подписи, потому что они включаются в идентификатор.

10.5. Новое поведение `OP_NOP5` в случае успеха должно точно соответствовать старому поведению `OP_NOP5`. То есть в случае успеха оно не должно влиять на стек.

10.6. Segwit-адресами являются **a** (p2wpkh) и **c** (p2wsh). **d** — это адрес p2sh, он может содержать в сценарии погашения вложенный платеж p2wpkh или p2wsh, но мы не можем утверждать этого. Единственное, что можно сказать наверняка, — это адрес p2sh, а не segwit-адрес.

10.7. Версия свидетеля используется для облегчения будущих обновлений. Здесь действует простое правило: неизвестные версии свидетелей принимаются. При развертывании новой версии свидетеля старые узлы будут принимать любой платеж, который расходует выход с этой новой версией свидетеля. Это позволяет избежать расщепления блокчейна на новые и старые узлы.

10.8. Все данные, имеющиеся в сценарии подписи, помещаются на стек. В данном случае в сценарии подписи таких элементов нет, поэтому он ничего не делает. Затем на стек будет помещен байт `00`, а затем `c805...cba8`. Программа на языке Script завершится, и узел проверит верхний элемент в стеке. Он не равен нулю, а значит, транзакция будет признана действительной.

10.9. Новый узел заметит, что выход соответствует шаблону segwit. Он также заметит, что версия свидетеля — `00`, а программа свидетеля имеет длину 20 байт. То есть это выход p2wpkh. Чтобы потратить такой выход, сценарий подписи должен быть пустым, а свидетель должен содержать подпись и открытый ключ, соответствующий программе свидетеля $РКН_{У}$. Шаблон p2wpkh заполняется с использованием подписи и открытого ключа из поля свидетеля и РКН из сценария открытого ключа (программа свидетеля). Заполненный шаблон затем обрабатывается как обычно.

10.10. Корень дерева Меркла комиссионных отчислений можно поместить в правую ветвь под обязательством. Но вам также необходимо поместить корень дерева Меркла комиссионных в структуру свидетеля для входа монетарной транзакции, чтобы старые segwit-узлы смогли проверить корневой хеш свидетеля.

10.11. Старый segwit-узел будет проверять блоки точно так же, как раньше. Зарезервированное значение свидетеля будет взято из свидетеля для входа монетарной транзакции. Хеш в свидетеле позволит старому узлу создать свидетельство и сравнить его с хешем в выходе `OP_RETURN`, но он не будет знать, что зарезервированное значение свидетеля является корнем дерева Меркла для комиссионных. То есть старые узлы не будут проверять дерево Меркла для комиссионных.

Новый узел выполнит ту же проверку, что и старый, а также вычислит корень Меркла для комиссионных и сравнит его с хешем в свидетеле монетарной транзакции.

Глава 11

11.1. Софт-форк ужесточает правила согласования. Это означает, что блоки, созданные новыми узлами, гарантированно будут приняты старыми узлами.

11.2. (а) Новая ветвь будет стерта старой ветвью.

(б) Потому что это произойдет *не сразу, но обязательно* произойдет, когда старая ветвь догонит новую и превзойдет ее. Для этого может потребоваться создать довольно много блоков, в зависимости от первоначального дефицита.

(в) В новую версию Биткоин можно добавить защиту от стирания — например, потребовав, чтобы первый блок после расщепления имел определенное свойство, делающее его недействительным в старой цепочке. Например, Bitcoin Cash требует, чтобы первый блок имел размер ≥1 000 000 байт.

11.3. Недолго, новая ветвь быстро превзойдет старую, и та будет стерта или реорганизована.

11.4. 2016 блоков. Состояние `LOCKED_IN` всегда длится один период ретаргетинга.

11.5. И те и другие. Старые узлы могут создавать блоки, недопустимые для новых узлов. Аналогично новые узлы могут создавать блоки, недопустимые для старых узлов.

11.6. Если новые узлы не будут обладать большей долей хешрейта, старые узлы могут вызывать длительное расщепление блокчейна. Фактически это может привести к появлению двух криптовалют.

11.7. Защита от повторения необходима потому, что транзакция, предназначенная для одной ветви, не должна оказаться в другой.

11.8. Да. Предположим, что 11 временных меток перед B_1 после сортировки по значению выстраиваются в последовательность

```
a ≤ b ≤ c ≤ d ≤ e ≤ MTP1 ≤ g ≤ h ≤ i ≤ j ≤ k
```

Чтобы вычислить MTP_2 блока B_2 после B_1, добавим T_1 в этот список. Поскольку временная метка блока должна быть строго позже, чем MTP (медианное время в прошлом) блока, время T_1 должно находиться справа от MTP_1 в списке. Например:

```
a ≤ b ≤ c ≤ d ≤ e ≤ MTP1 ≤ g ≤ h ≤ T1 ≤ i ≤ j ≤ k
```

Вам также нужно удалить из списка метку времени блока с наименьшей высотой. Независимо от того, какая метка будет удалена, MTP_2 будет либо равно MTP_1 (если удалить временную метку справа), либо окажется меткой времени непосредственно справа от MTP_1 (если удалить метку слева), которая может быть меткой g или T_1:

Если $MTP_2 = MTP_1$, тогда MTP2 < тайм-аут, потому что MTP_1 < тайм-аут.

Если $MTP_2 = g$, тогда $MTP_2 \leq T_1$ < тайм-аут.

Если $MTP_2 = T_1$, тогда MTP_2 < тайм-аут, потому что T_1 < тайм-аут.

То есть MTP блока B_2 меньше чем тайм-аут во всех случаях и все блоки (>95%) в числе последних 2016 блоков сигнализируют о поддержке, а это означает переход в состояние `LOCKED_IN` и, 2016 блоков спустя, переход в состояние `ACTIVE`.

11.9. Часть (<30%) пользователей начнет отвергать блоки, не соответствующие новым требованиям. Это вызовет расщепление блокчейна, которое будет длиться, пока майнеры, поддерживающие старую ветвь, будут оставаться в большинстве.

11.10. Когда большая часть пользователей начнет отвергать старые блоки, майнеры наверняка не захотят добывать старые блоки, потому что вознаграждение за них станет практически бесполезным. Майнерам трудно будет продать свои старые монеты на бирже или оплатить ими счет за электри-

чество. Напротив, если они переключатся на добычу новых блоков, у них будет больше вариантов обменять вознаграждение на товары, услуги или другие валюты.

11.11. Пользователи, не занимающиеся майнингом и использующие старое программное обеспечение, автоматически переключатся на новую ветвь, как только она станет сильнее старой. Это связано с тем, что в софт-форке новая ветвь продолжает оставаться действительной с точки зрения старого программного обеспечения.

B Веб-ресурсы

1. Сатоши Накамото (Satoshi Nakamoto), «Bitcoin: A Peer-to-Peer Electronic Cash System», 2008, http://mng.bz/lppR.[1]
2. Bitcoin Stack Exchange, http://mng.bz/BDDl.
3. Руководство разработчика Биткоин, http://mng.bz/dPP1.
4. Репозиторий с исходным кодом Bitcoin Core, http://mng.bz/rBBj.
5. Исходный код для книги Калле Розенбаума (Kalle Rosenbaum) «Грокаем технологию Биткоин», http://mng.bz/qBO2.
6. «Financial Inclusion Data/Global Findex», The World Bank, 2014, http://mng.bz/Vqqx.
7. Статья в Википедии «Reception of WikiLeaks: Response from the financial industry», http://mng.bz/gYnV.
8. Менелаос Хаджикостис (Menelaos Hadjicostis), «Bank of Cyprus Depositors Lose 47.5% of Savings», USA Today, 29 июля, 2013, http://mng.bz/pEez.
9. Предложения по улучшению Биткоин (Bitcoin Improvement Proposal, BIP), GitHub, http://mng.bz/OA0E.
10. Страница, предлагающая на выбор разные кошельки Биткоин, http://mng.bz/xJJ6.

[1] Перевод на русский язык: Сатоши Накамото, «Биткоин: система цифровой пиринговой наличности», https://bitcoin.org/files/bitcoin-paper/bitcoin_ru.pdf. — *Примеч. пер.*

11. Андреа Корбеллини (Andrea Corbellini), «Elliptic Curve Point Addition», http://mng.bz/YOBA.
12. Обсуждение причин использования двойного хеширования SHA256 в Биткоин, Stack Exchange Cryptography, http://mng.bz/G2aO.
13. Полный список операторов языка Script в Биткоин, Bitcoin Wiki, http://mng.bz/A22Q.
14. Артур Жерве (Arthur Gervais), Гассан О. Караме (Ghassan O. Karame), Дамиан Грубер (Damian Gruber) и Срджан Капкун (Srdjan Capkun), «On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients», Cryptology ePrint Archive, http://mng.bz/ZZZ9.
15. Питер Тодд (Peter Todd), «What Attack Does the Difficulty Drop Rate Limiter Prevent?», http://mng.bz/zgWQ.
16. Lightning Labs, Lightning Resources, http://mng.bz/RGGa.
17. Обозреватель блокчейна, позволяющий исследовать транзакции, как обсуждалось в главе 9, http://mng.bz/J88Q.
18. Инструкция по запуску полного узла, http://mng.bz/2AAw.
19. Страница загрузки Bitcoin Core, http://mng.bz/177R.
20. Инструкция по началу использования Биткоин, http://mng.bz/P885.