O'REILLY®

Осваиваем Биткойн

Программирование открытого блокчейна

Третье издание

Андреас М. Антонопулос
Дэвид А. Хардинг

Андреас М. Антонопулос и Дэвид А. Хардинг

Осваиваем Биткойн

Andreas M. Antonopoulos and David A. Harding

Mastering Bitcoin

Programming the Open Blockchain

THIRD EDITION

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Андреас М. Антонопулос и Дэвид А. Хардинг

Осваиваем Биткойн

Программирование открытого блокчейна

ТРЕТЬЕ ИЗДАНИЕ

BOOKS.KZ
2024

УДК 004.738.5:336.74Bitcoin
ББК 32.971.35+65.262.6с
А72

Антонопулос А. М., Хардинг Д. А.
А72 Осваиваем Биткойн. 3-е изд. / пер. с анг. В. И. Бахура. – М.: Book.kz, 2024. – 386 с.: ил.

ISBN 978-6-01810-344-5

Третье издание бестселлера включает подробное введение в самую известную криптовалюту – биткойн, а также в лежащую в ее основе технологию блокчейн. Приведено описание технических основ биткойна и других валют, описание децентрализованной сети Биткойн, пиринговой архитектуры, жизненного цикла транзакций и принципов обеспечения безопасности. Показаны методики разработки блокчейн-приложений с многочисленными примерами кода.

Книга будет интересна разработчикам, инженерам, архитекторам программных и прочих систем, а также всем, кто хочет глубже узнать о криптовалютах и блокчейн-технологиях.

УДК 004.738.5:336.74Bitcoin
ББК 32.971.35+65.262.6с

ISBN 978-1-098-15009-9 (анг.)
ISBN 978-6-01810-344-5 (казах.)

Посвящается моей маме Терезе (1946–2017).
Она научила меня любить книги
и не принимать на веру мнение авторитетов.
Спасибо, мама.
– Андреас

Для Аманды.
Только после встречи с тобой я стал жить в раю.
– Дейв

Содержание

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Отзывы о книге «Осваиваем Биткойн», третье издание

«Книга “Осваиваем Биткойн” очень полезна для каждого, кому хочется или требуется понять технологию биткойна и концепции его протоколов».

– Рене Пикхардт (René Pickhardt), разработчик Bitcoin Lightning Network

«Окунитесь в увлекательный мир Биткойна с книгой “Осваиваем Биткойн”. Это надежное пособие, которое поможет разобраться в тонкостях данной цифровой валюты. Если вы разработчик, инвестор или просто интересуетесь будущим денег, эта содержательная книга станет для вас путеводителем, который предоставит необходимые знания и поможет чувствовать себя уверенно в эпоху интернета денег».

– Жорж Лесмес (Jorge Lesmes), старший директор NTT DATA

«Спустя почти десятилетие после выхода в свет третье издание книги “Осваиваем Биткойн” закрепило за ней роль основного источника технического образовательного контента по теме биткойна. Ни одна другая книга не является столь же всеобъемлющей и актуальной».

– Олаолува Осунтокун (Olaoluwa Osuntokun), технический директор Lightning Labs

«Всесторонний обзор всего, что происходит в системе Биткойн, и как все это сочетается друг с другом».

– Марк «Мурч» Эрхардт (Mark «Murch» Erhardt), биткойн-инженер Chaincode Labs

Предисловие

Как была написана книга о Биткойне

Впервые я (Андреас) узнал о Биткойне в середине 2011 года. Первой моей реакцией было что-то вроде «Ха! Деньги умников!», и еще полгода я просто игнорировал это явление, не понимая его важности. Некоторым утешением может служить тот факт, что подобную реакцию можно было наблюдать у многих моих умнейших знакомых. Во второй раз я столкнулся с Биткойном в одном из обсуждений списка рассылки. Тогда я решил прочитать техническое описание, написанное Сатоши Накамото (Satoshi Nakamoto), и разобраться в сути дела. До сих пор помню момент, когда я закончил читать эти девять страниц и осознал, что биткойн – это не просто цифровая валюта, а доверительная сеть, способная стать основой для многого другого, помимо собственно денег. С осознанием факта, что «это не столько деньги, сколько децентрализованная доверительная сеть», я приступил к четырехмесячному исследованию Биткойна и начал впитывать каждый крохотный обрывок информации о нем, который только мог найти. На волне энтузиазма я проводил по 12 и более часов в день у экрана, читая, записывая, кодируя и изучая как можно больше. Я вышел из этого погружения похудевшим более чем на 20 фунтов (примерно 9 кг) из-за неправильного питания, с твердым намерением посвятить себя дальнейшей работе над Биткойном.

Спустя два года, после создания нескольких небольших стартапов для изучения различных услуг и продуктов на основе биткойна, я понял, что настало время написать свою первую книгу. Биткойн оказался той темой, которая вызвала у меня бурный творческий подъем и поглотила все мои мысли; это самая захватывающая технология, с которой я столкнулся после появления интернета. Настало время поделиться своей страстью к этой удивительной технологии с широким кругом читателей.

Целевая аудитория

Эта книга рассчитана в основном на программистов. Если вы умеете пользоваться языком программирования, эта книга научит вас работать с криптовалютами, использовать их и разрабатывать программное обеспечение для работы с ними. Первые несколько глав также можно использовать в качестве углубленного введения в биткойн для тех, кто не является кодером и пытается понять внутреннее устройство Биткойна и криптовалют.

Почему на обложке букашки?

Муравей-листорез – это биологический вид, который обладает очень сложным поведением в составе колонии-суперорганизма. При этом каждый отдельный муравей живет в соответствии с набором простых правил, обусловленных социальным взаимодействием и обменом химическими запахами (феромонами). По данным Википедии: «Как и люди, муравьи-листорезы образуют самые большие и сложные социальные живые сообщества на Земле». В действительности муравьи-листорезы не едят листья, а используют их для выращивания грибка в качестве основного источника пищи для обитателей своей колонии. Представляете? Эти муравьи занимаются сельским хозяйством!

Хотя муравьи живут в кастовом обществе и имеют королеву для производства потомства, в их колонии нет центральной власти или лидера. Высокоинтеллектуальное и совершенное поведение многомиллионной колонии – это эмерджентное свойство, возникающее в процессе взаимодействия особей внутри социальной сети.

Природа наглядно показывает: децентрализованные системы могут быть жизнеспособными. Они могут формировать эмерджентную сложность и невероятную развитость без необходимости в центральном руководстве, иерархии или сложных элементах.

Биткойн – это чрезвычайно сложная децентрализованная доверительная сеть, которая способна поддерживать огромное количество финансовых процессов. При этом каждый узел в сети Биткойн соблюдает всего лишь несколько простых правил. Взаимодействие между множеством узлов – именно это обеспечивает формирование такого продуманного поведения, а не сложность или доверие к какому-либо отдельному узлу. Подобно муравейнику, сеть Биткойн – это жизнестойкая система простых узлов, подчиняющихся простым правилам. Вместе они способны совершать удивительные вещи без какой-либо централизованной координации.

Условные обозначения, используемые в этой книге

В этой книге используются следующие типографские нормы:

Курсив
: Обозначает новые термины, URL-адреса, адреса электронной почты, имена и расширения файлов.

`Моноширинный шрифт`
: Используется в листингах программ, а также внутри абзацев для обозначения таких элементов программы, как имена переменных или функций, базы данных, типы данных, переменные среды, операторы и ключевые слова.

`Моноширинный полужирный шрифт`
: Показывает команды или другой текст, который должен быть набран пользователем вручную.

Моноширинный курсив
Обозначает текст для замены на значения, вводимые пользователем, или на значения из контекста.


Этот элемент обозначает совет или рекомендацию.


Этот элемент обозначает общее примечание.


Этот элемент обозначает предупреждение или оповещение о потенциальной опасности.

Примеры кода

Все фрагменты кода могут быть воспроизведены на большинстве операционных систем с минимальным набором компиляторов и интерпретаторов для соответствующих языков. В случае необходимости приводятся базовые инструкции по установке и пошаговые примеры вывода этих инструкций.

Некоторые фрагменты кода и его выходные результаты были переформатированы для печати. Во всех этих случаях строки разделены символом обратной косой черты (\), за которым следует символ новой строки. При расшифровке примеров следует удалить эти два символа и снова соединить строки, чтобы получить идентичный результат, как показано в примере.

Во всех фрагментах кода по возможности используются реальные значения и вычисления, так что можно переходить от примера к примеру и видеть те же результаты в любом коде, который вы будете писать для вычисления тех же значений.

Использование примеров кода

Эта книга призвана помочь вам в вашей работе. Как правило, если в книге предлагаются примеры кода, их можно использовать в своих программах и документации. Вам не нужно обращаться за разрешением, если только вы не воспроизводите значительную часть кода. Например, написание программы, использующей несколько фрагментов кода из этой книги, не требует разрешения. Продажа или распространение примеров из книг O'Reilly требует разрешения. Для ответа на вопрос со ссылкой на эту книгу и цитированием примера кода разрешение не требуется. Включение значительного количества примеров из этой книги в документацию вашего продукта требует разрешения.

Мы ценим, но не требуем указания авторства. Авторство обычно включает название, автора, издателя и ISBN. Например: «Mastering Bitcoin, 3rd ed., by Andreas M. Antonopoulos and David A. Harding (O'Reilly). Copyright 2024 David Harding, ISBN 978-1-098-15009-9».

Некоторые издания этой книги распространяются под лицензией с открытым исходным кодом, например CC-BYNC, и в этом случае применяются условия этой лицензии.

Если вы считаете, что использование примеров кода выходит за рамки добросовестного использования или вышеуказанного разрешения, свяжитесь с нами по адресу permissions@oreilly.com.

Изменения относительно предыдущего издания книги

Особое внимание в третьем издании уделено переработке текста второго издания 2017 года и сохранившегося текста первого издания 2014 года. Кроме того, было добавлено множество концепций, актуальных для разработки современного Биткойна в 2023 году:

Глава 4
: Мы перегруппировали информацию об адресах таким образом, чтобы работать со всеми данными в историческом порядке. Мы добавили новый раздел по P2PK (где под «адресом» подразумевался IP-адрес), обновили предыдущие разделы по P2PKH и P2SH, а также добавили новые разделы по segwit/bech32 и taproot/bech32m.

Прежние главы 6 и 7
: Структура глав 6 «Транзакции» и 7 «Расширенные транзакции» была переработана и дополнена. Теперь это четыре новые главы: глава 6 «Транзакции» (структура транзакций), глава 7 «Авторизация и аутентификация», глава 8 «Цифровые подписи» и глава 9 «Плата за транзакции».

Глава 6
: Мы добавили почти полностью новую версию текста, описывающего структуру транзакций.

Глава 7
: Мы добавили новый текст о MAST, P2C, мультиподписях без скриптов, а также о taproot и tapscript.

Глава 8
: Мы пересмотрели текст о ECDSA и добавили новое описание подписей Шнорра, мультиподписей и пороговых подписей.

Глава 9
: Мы добавили почти полностью новый текст о комиссионных сборах, функциях повышения комиссионных сборов RBF и CPFP, пиннинге транзакций, ретрансляции пакетов и политике исключений для CPFP.

Глава 10
: Мы добавили текст о передаче компактных блоков, существенно обновили описание фильтров bloom с лучшим описанием проблем конфиденциальности, а также написали новый текст о компактных блочных фильтрах.

Глава 11
: Добавлен текст о технологии Signet.

Глава 12

Добавлен текст о BIP8 и коде активации Speedy Trial.

Приложения

Мы удалили специфические библиотечные приложения. После приложения с оригинальным техническим описанием мы добавили новое приложение с описанием того, чем реализация и свойства биткойна отличаются от предложенных в техническом описании.

Адреса и транзакции Биткойна в этой книге

Биткойн-адреса, транзакции, ключи, QR-коды и данные блокчейна, используемые в этой книге, по большей части реальны. Это означает, что вы можете просматривать блокчейн, изучать транзакции, приведенные в качестве примеров, извлекать их с помощью собственных скриптов или программ и т. д.

Однако имейте в виду, что закрытые ключи для построения адресов либо напечатаны в книге, либо «сожжены». Это означает, что если вы отправите деньги на любой из этих адресов, они будут либо навсегда потеряны, либо, как вариант, любой читатель книги сможет забрать их, используя напечатанные в ней закрытые ключи.

НЕ ОТПРАВЛЯЙТЕ ДЕНЬГИ НИ ПО ОДНОМУ ИЗ АДРЕСОВ, УКАЗАННЫХ В ЭТОЙ КНИГЕ. Ваши деньги достанутся другому читателю или будут утеряны навсегда.

Как связаться с авторами

Обратиться к Андреасу М. Антонопулосу можно через его персональный сайт: https://antonopoulos.com.

Подпишитесь на Андреаса в Facebook: https://facebook.com/AndreasMAntonopoulos.

Подпишитесь на Андреаса в Twitter: https://twitter.com/aantonop.

Подпишитесь на Андреаса в LinkedIn: https://linkedin.com/company/aantonop.

Большое спасибо всем спонсорам Андреаса, которые поддерживают его работу ежемесячными пожертвованиями. Вы можете посмотреть его страницу на Patreon здесь: https://patreon.com/aantonop.

Информация о книге «Осваиваем Биткойн», а также об открытом издании и переводах Андреаса доступна на сайте https://bitcoinbook.info.

Связаться с Дэвидом А. Хардингом можно через его личный сайт: https://dtrt.org.

Благодарности за первое и второе издания

Андреас М. Антонопулос

Эта книга является результатом труда и усилий многих людей. Я благодарен за помощь, которую мне оказали друзья, коллеги и даже совершенно незнакомые

люди, которые присоединились ко мне в стремлении написать исчерпывающе полную техническую книгу о криптовалютах и Биткойне.

Как невозможно разграничить технологию Биткойн и сообщество биткойна, так и это издание является в равной степени продуктом этого сообщества и книгой о технологии. С самого начала и до самого конца моя работа над этой книгой получала ободрение, поддержку и благодарность от всего биткойн-сообщества. Более того, эта книга позволила мне в течение двух лет оставаться частью замечательного сообщества, и я не могу не поблагодарить вас за то, что вы приняли меня в него. Слишком много людей, чтобы называть их по именам, – тех, кого я встречал на конференциях, мероприятиях, семинарах, сборах, посиделках в пиццерии и небольших личных встречах, а также многих, кто общался со мной в Twitter, на reddit, на bitcointalk.org и на GitHub, – всех, кто оказал влияние на эту книгу. Каждая идея, сравнение, вопрос, ответ и объяснение, которые вы найдете в этой книге, были в какой-то момент поддержаны, проверены или улучшены благодаря общению с сообществом. Спасибо всем вам за поддержку; без вас эта книга не увидела бы свет. Я бесконечно вам благодарен.

Разумеется, путь автора начинается задолго до выхода первой книги. Моим родным языком (и языком обучения в школе) был греческий, поэтому на первом курсе университета мне пришлось посещать коррекционный курс английского языка. Я благодарен Диане Кордас (Diana Kordas), моему преподавателю английского, которая помогала мне в тот год обрести уверенность и навыки. Позже, став профессионалом, я совершенствовал свои навыки написания технических статей по тематике центров обработки данных, работая в журнале Network World. Я благодарен Джону Диксу (John Dix) и Джону Галланту (John Gallant), которые предоставили мне первую работу в качестве обозревателя в Network World, а также моему редактору Майклу Куни (Michael Cooney) и моей коллеге Джоне Тилл Джонсон (Johna Till Johnson), которые занимались правкой моих колонок и делали их пригодными для публикации. Написание 500 слов в неделю в течение четырех лет позволило мне получить достаточно опыта, чтобы в конце концов задуматься об авторстве.

Также выражаю благодарность тем, кто поддержал меня во время подачи заявки на издание книги в O'Reilly своими рекомендациями и рецензиями. В частности, спасибо Джону Галланту (John Gallant), Грегори Нессу (Gregory Ness), Ричарду Стиннону (Richard Stiennon), Джоэлу Снайдеру (Joel Snyder), Адаму Левину (Adam B. Levine), Сандре Гитлен (Sandra Gittlen), Джону Диксу (John Dix), Джоне Тилл Джонсон (Johna Till Johnson), Роджеру Веру (Roger Ver) и Джону Матонису (Jon Matonis). Особая благодарность Ричарду Кагану (Richard Kagan) и Тимону Маттошко (Tymon Mattoszko), которые рецензировали ранние версии заявки, а также Мэтью Тейлору (Matthew Taylor), который выполнил техническое редактирование и форматирование текста заявки.

Благодарю Крикета Лю (Cricket Liu), автора книги «*DNS и BIND*» в O'Reilly, который познакомил меня с издательством. Спасибо также Майклу Лоукидесу (Michael Loukides) и Эллисон Макдональд (Allyson MacDonald) из O'Reilly, которые в течение нескольких месяцев работали над этой книгой. Эллисон была особенно терпелива в моменты срыва сроков и задержек с выпуском, когда жизнь вносила коррективы в запланированный нами график. Выражаю благодарность Тимоти Макговерну (Timothy McGovern) за руководство работами

при выпуске второго издания, Ким Кофер (Kim Cofer) за терпеливую редактуру, а также Ребекке Панцер (Rebecca Panzer) за иллюстрации к многочисленным новым рисункам.

Самыми трудными были несколько черновиков первых глав, потому что тема Биткойна является сложной для последовательного изучения. Каждый раз, когда я дергал за одну из нитей технологии Биткойн, мне приходилось распутывать всю тему. То и дело я останавливался и впадал в уныние в поисках простого объяснения темы и понятного изложения такого сложного технического материала. Я благодарен своему другу и наставнику Ричарду Кагану (Richard Kagan), который помог разобраться с историей и преодолеть моменты писательского кризиса. Выражаю благодарность разработчикам из группы San Francisco Bitcoin Developers Meetup, а также Таарику Льюису (Taariq Lewis) и Дениз Терри (Denise Terry) за помощь в проверке предварительных версий. Также выражаю особую благодарность Эндрю Науглеру (Andrew Naugler) за дизайн инфографики.

В процессе работы над книгой я выложил первые черновики на GitHub и предложил аудитории высказать свои замечания. Было получено более сотни комментариев, предложений, исправлений и дополнений. Этот вклад и моя благодарность за него подробно описаны в разделе «Черновик ранней версии (вклад GitHub)». В первую очередь я искренне благодарен моим добровольным редакторам на GitHub Мингу Т. Нгуену (Ming T. Nguyen, 1-е издание) и Уиллу Биннсу (Will Binns, 2-е издание), которые без устали курировали, принимали и обрабатывали предложения, сообщения о проблемах, а также исправляли ошибки на GitHub.

После завершения работы над книгой она прошла несколько этапов технического рецензирования. Благодарю Крикета Лю (Cricket Liu) и Лорна Ланца (Lorne Lantz) за их тщательную проверку, комментарии и поддержку.

Несколько разработчиков биткойна предоставили примеры кода, обзоры, комментарии и полезные советы. Спасибо Амиру Тааки (Amir Taaki) и Эрику Воскуилу (Eric Voskuil) за примеры кода и множество замечательных комментариев; Крису Клешульте (Chris Kleeschulte) за информацию о Bitcore; Виталику Бутерину (Vitalik Buterin) и Ричарду Киссу (Richard Kiss) за помощь с расчетами эллиптических кривых и вклад в код; Гэвину Андресену (Gavin Andresen) за исправления, комментарии и поддержку; Михалису Каргакису (Michalis Kargakis) за комментарии, вклад и запись btcd; а также Робину Инге (Robin Inge) за исправления, внесенные в улучшение второго издания. При подготовке второго издания мне вновь оказали значительную поддержку многие разработчики Bitcoin Core, в том числе Эрик Ломброзо (Eric Lombrozo), который раскрыл суть сегрегированного свидетельства, Люк Дашжр (Luke Dashjr), который помог улучшить главу о транзакциях, Джонсон Лау (Johnson Lau), который рецензировал сегрегированное свидетельство и другие главы, и многие другие. Я благодарен Джозефу Пуну (Joseph Poon), Тадге Драйе (Tadge Dryja) и Олаолуве Осунтокуну (Olaoluwa Osuntokun), которые объясняли суть Lightning Network, рецензировали мои работы и отвечали на вопросы, когда я оказывался в затруднительном положении.

Своей любовью к словесности и книгам я обязан моей маме Терезе, которая вырастила меня в доме, где все стены были уставлены книгами. Мама также ку-

пила мне мой первый компьютер в 1982 году, хотя я сам считал себя технофобом. Мой отец Менелаос, инженер-строитель, который недавно опубликовал свою первую книгу в возрасте 80 лет, был тем, кто научил меня логическому и аналитическому мышлению, а также любви к науке и инженерии.

Спасибо всем, кто поддерживал меня на всех этапах этого пути.

Благодарности за третье издание

Дэвид А. Хардинг

Знакомство с неинтерактивным протоколом подписи Шнорра, которое начинается с описания интерактивного протокола идентификации Шнорра в разделе «Подписи Шнорра», было написано под сильным впечатлением от введения в эту тему в книге «Подписи Борроммеанского кольца» (Borrommean Ring Signatures, 2015), написанной Грегори Максвеллом (Gregory Maxwell) и Эндрю Поэлстрой (Andrew Poelstra). Я глубоко признателен каждому из них за всю их бескорыстную помощь в течение последнего десятилетия.

Неоценимую техническую помощь в работе над черновиками этой книги оказали Хорхе Лесмес (Jorge Lesmes), Олаолува Осунтокун (Olaoluwa Osuntokun), Рене Пикхардт (René Pickhardt) и Марк Эрхардт (Mark «Murch» Erhardt). В частности, невероятно глубокая и содержательная рецензия Мерча, как и его готовность оценить несколько итераций одного и того же текста, позволили поднять качество этой книги выше моих самых смелых ожиданий.

Я также выражаю глубокую благодарность Джимми Сонгу (Jimmy Song) за его предложение принять участие в этом проекте, моему соавтору Андреасу (Andreas) за разрешение обновить его бестселлер, Анджеле Руфино (Angela Rufino) за сопровождение моего авторского процесса в O'Reilly, а также всем остальным сотрудникам O'Reilly, благодаря которым работа над третьим изданием стала приятным и продуктивным опытом.

В заключение, я просто не представляю, как можно выразить благодарность всем участникам проекта Bitcoin, которые помогали мне в моей жизни – от создания программ, которые я использую, до обучения их работе и помощи в передаче тех крох знаний, которые я приобрел. Вас очень много, и я не смогу перечислить все ваши имена, но я часто думаю о вас и знаю, что мой вклад в эту книгу был бы невозможен без всей вашей помощи.

Черновик ранней версии (вклад GitHub)

Многие авторы предложили свои комментарии, исправления и дополнения к черновому варианту ранней версии, размещенной на GitHub. Спасибо всем за ваш вклад в создание этой книги.

Ниже приведен список известных участников GitHub, в скобках указан их учетный идентификатор GitHub:

- Abdussamad Abdurrazzaq (AbdussamadA);
- Adán SDPC (aesedepece);
- Akira Chiku (achiku);
- Alex Waters (alexwaters);

- Andrew Donald Kennedy (grkvlt);
- Andrey Esaulov (andremaha);
- andronoob;
- AnejaBK;
- Appaji (CITIZENDOT);
- ariesunny;
- Arthur O'Dwyer (Quuxplusone);
- bargitta;
- Basem Alasi (Bamskki);
- bisqfan;
- bitcoinctf;
- blip151;
- Bryan Gmyrek (physicsdude);
- Carlos Sims (simsbluebox);
- Casey Flynn (cflynn07);
- cclauss;
- Chapman Shoop (belovachap);
- chrisd95;
- Christie D'Anna (avocadobreath);
- Cihat Imamoglu (cihati);
- Cody Scott (Siecje);
- coinradar;
- Cragin Godley (cgodley);
- Craig Dodd (cdodd);
- dallyshalla;
- Dan Nolan (Dan-Nolan);
- Dan Raviv (danra);
- Darius Kramer (dkrmr);
- Darko Janković (trulex);
- David Huie (DavidHuie);
- didongke;
- Diego Viola (diegoviola);
- Dimitris Tsapakidis (dimitris-t);
- Dirk Jäckel (biafra23);
- Dmitry Marakasov (AMDmi3);
- drakos (Jolly-Pirate);
- drstrangeM;
- Ed Eykholt (edeykholt);
- Ed Leafe (EdLeafe);
- Edward Posnak (edposnak);
- Elias Rodrigues (elias19r);
- Eric Voskuil (evoskuil);
- Eric Winchell (winchell);
- Erik Wahlström (erikwam);
- effectsToCause (vericoin);
- Esteban Ordano (eordano);
- ethers;
- Evlix;
- fabienhinault;
- Fan (whiteath);
- Felix Filozov (ffilozov);
- Francis Ballares (fballares);
- François Wirion (wirion);
- Frank Höger (francyi);
- Gabriel Montes (gabmontes);
- Gaurav Rana (bitcoinsSG);
- genjix;
- Geremia;
- Gerry Smith (Hermetic);
- gmr81;
- Greg (in3rsha);
- Gregory Trubetskoy (grisha);
- Gus (netpoe);
- halseth;
- harelw;
- Harry Moreno (morenoh149);
- Hennadii Stepanov (hebasto);
- Holger Schinzel (schinzelh);
- Ioannis Cherouvim (cherouvim);
- Ish Ot Jr. (ishotjr);
- ivangreene;
- James Addison (jayaddison);
- Jameson Lopp (jlopp);
- Jason Bisterfeldt (jbisterfeldt);
- Javier Rojas (fjrojasgarcia);
- Jordan Baczuk (JBaczuk);
- Jeremy Bokobza (bokobza);
- JerJohn15;
- jerzybrzoska;
- Jimmy DeSilva (jimmydesilva);
- Jo Wo (jowo-io);
- Joe Bauers (joebauers);
- joflynn;
- Johnson Lau (jl2012);
- Jonathan Cross (jonathancross);
- Jorgeminator;
- jwbats;
- Kai Bakker (kaibakker);
- kollokollo;
- krupawan5618;
- kynnjo;
- Liangzx;
- lightningnetworkstores;
- lilianrambu;

- Liu Yue (lyhistory);
- Lobbelt;
- Lucas Betschart (lclc);
- Matt Wesley (MatthewWesley);
- Magomed Aliev (30mb1);
- Mai-Hsuan Chia (mhchia);
- Marco Falke (MarcoFalke);
- María Martín (mmartinbar);
- Marcus Kiisa (mkiisa);
- Mark Erhardt (Xekyo);
- Mark Pors (pors);
- Martin Harrigan (harrigan);
- Martin Vseticka (MartyIX);
- Marzig (marzig76);
- Matt McGivney (mattmcgiv);
- Matthijs Roelink (Matthiti);
- Maximilian Reichel (phramz);
- MG-ng (MG-ng);
- Michalis Kargakis (kargakis);
- Michael C. Ippolito (michaelcippolito);
- Michael Galero (mikong);
- Michael Newman (michaelbnewman);
- Mihail Russu (MihailRussu);
- mikew (mikew);
- milansismanovic;
- Minh T. Nguyen (enderminh);
- montvid;
- Morfies (morfies);
- Nagaraj Hubli (nagarajhubli);
- Nekomata (nekomata-3);
- nekonenene;
- Nhan Vu (jobnomade);
- Nicholas Chen (nickycutesc);
- Ning Shang (syncom);
- Oge Nnadi (ogennadi);
- Oliver Maerz (OliverMaerz);
- Omar Boukli-Hacene (oboukli);
- Óscar Nájera (Titan-C);
- Parzival (Parz-val);
- Paul Desmond Parker (sunwukonga);
- Philipp Gille (philippgille);
- ratijas;
- rating89us;
- Raul Siles (raulsiles);
- Reproducibility Matters (TheCharlatan);
- Reuben Thomas (rrthomas);
- Robert Furse (Rfurse);
- Roberto Mannai (robermann);
- Richard Kiss (richardkiss);
- rszheng;
- Ruben Alexander (hizzvizz);
- Sam Ritchie (sritchie);
- Samir Sadek (netsamir);
- Sandro Conforto (sandroconforto);
- Sanjay Sanathanan (sanjays95);
- Sebastian Falbesoner (theStack);
- Sergei Tikhomirov (s-tikhomirov);
- Sergej Kotliar (ziggamon);
- Seiichi Uchida (topecongiro);
- shaysw;
- Simon de la Rouviere (simondlr);
- simone-cominato;
- sindhoor7;
- Stacie (staciewaleyko);
- Stephan Oeste (Emzy);
- Stéphane Roche (Janaka-Steph);
- takaya-imai;
- Thiago Arrais (thiagoarrais);
- Thomas Kerin (afk11);
- Tochi Obudulu (tochicool);
- Tosin (tkuye);
- Vasil Dimov (vasild);
- venzen;
- Vlad Stan (motorina0);
- Vijay Chavda (VijayChavda);
- Vincent Déniel (vincentdnl);
- weinim;
- wenxiaolong (QingShiLuoGu);
- wenzhenxiang;
- Will Binns (wbnns);
- wintercooled;
- wjx;
- wll2007;
- Wojciech Langiewicz (wlk);
- Yancy Ribbens (yancyribbens);
- yjjnls;
- Yoshimasa Tanabe (emag);
- yuntai;
- yurigeorgiev4;
- Zheng Jia (zhengjia);
- Zhou Liang (zhouguoguo).

Об авторах

Андреас М. Антонопулос (Andreas M. Antonopoulos) – известный технический специалист и опытный предприниматель, который стал одним из самых известных и уважаемых представителей сферы Биткойн. Будучи прекрасным оратором, преподавателем и писателем, Андреас излагает сложные темы доступным и понятным языком. В качестве консультанта он оказывает помощь стартапам в определении, оценке и управлении рисками в сфере безопасности и бизнеса.

Андреас вырос вместе с интернетом, создав свою первую компанию, раннюю электронную доску объявлений (Bulletin Board System, BBS) и систему доступа в интернет, еще будучи подростком в своем доме в Греции. Он получил степень в области компьютерных наук, передачи данных и распределенных систем в Университетском колледже Лондона (University College London, UCL), который недавно вошел в десятку лучших университетов мира. После переезда в США Андреас стал соучредителем и руководителем успешной исследовательской компании в области технологий, а также консультантом десятков руководителей компаний из списка Fortune 500 по вопросам сетевых технологий, безопасности, центров обработки данных и облачных вычислений. Более двухсот его статей по вопросам безопасности, облачных вычислений и центров обработки данных были опубликованы в печати и распространены по всему миру. Он имеет два патента в области сетевых технологий и безопасности.

В 1990 году Андреас занялся преподаванием различных дисциплин в области ИТ в частной, профессиональной и академической среде. Он оттачивал свое ораторское мастерство перед аудиториями самого разного размера – от пяти руководителей в зале заседаний до тысяч человек на крупных конференциях. За его плечами более четырехсот выступлений. Он считается харизматичным оратором и преподавателем мирового класса. В 2014 году он был назначен преподавателем Университета Никосии – первого в мире университета, предлагающего степень магистра в области цифровых валют. В этой роли он помогал разрабатывать учебную программу и преподавал курс «Введение в цифровые валюты», предлагаемый в качестве массового открытого онлайн-курса в университете.

Будучи предпринимателем в сфере Биткойн, Андреас основал несколько биткойн-компаний и запустил несколько проектов с открытым исходным кодом. Он выступает в качестве советника нескольких биткойн- и криптовалютных компаний. Он широко публикуется в журналах и блогах, посвященных Биткойну, является постоянным ведущим популярного подкаста Let's Talk Bitcoin, а также часто выступает на конференциях по технологиям и безопасности по всему миру.

Дэвид А. Хардинг (David A. Harding) – технический писатель, который специализируется на создании документации для программного обеспечения с открытым исходным кодом. Он является соавтором еженедельного новостного бюллетеня *Bitcoin Optech* (2018–2023), учебников *21.co Bitcoin Computer* (2015–2017) и документации для разработчиков Bitcoin.org (2014–2015). Он также является членом грантового комитета Brink.dev (2022–2023) и бывшим членом его правления (2020–2022).

Колофон

Животное на обложке книги «Осваиваем Биткойн» – это муравей-листорез (Atta colombica). Муравей-листорез (простое название) – это зонтичный муравей из тропиков Южной и Центральной Америки, а также Мексики и юга Соединенных Штатов. Как и люди, муравьи-листорезы образуют самые большие и сложные сообщества животных на планете. Они получили свое название за то, что жуют листья, которые служат питанием в их грибковом хозяйстве.

Крылатые муравьи – как самцы, так и самки совершают массовый выход из гнезда для брачного полета, известного как ревоада (revoada). Самки спариваются с несколькими самцами, чтобы собрать 300 млн сперматозоидов, необходимых для создания колонии. Самки также хранят кусочки мицелия родительского грибного сада в инфрабуккальном кармане, расположенном в ротовой полости; они будут использовать его для создания собственных грибных садов. Опустившись на землю, самка теряет крылья и устраивает подземное гнездо для своей колонии. Успех новых королев невелик: лишь 2,5 % создают долгоживущую колонию.

Когда колония созревает, муравьи делятся на касты в зависимости от размера, каждая из которых выполняет различные функции. Обычно выделяют четыре касты: минимы – самые маленькие рабочие, которые ухаживают за молодняком и грибными садами; миноры, чуть крупнее минимов, являются первой линией обороны колонии, патрулируют окрестности и нападают на врагов; медиа – общие фуражиры, которые обрывают листья и приносят их фрагменты в гнездо; и майоры – самые крупные рабочие муравьи, которые действуют как солдаты, защищая гнездо от вторжения. Недавние исследования показали, что майоры также расчищают основные кормовые тропы и переносят громоздкие предметы обратно в гнездо.

Многие из животных, обитающих на обложках книг издательства O’Reilly, находятся под угрозой исчезновения; все они важны для мира.

Иллюстрация для обложки выполнена Карен Монтгомери (Karen Montgomery) на основе изображения из книги «Зарубежные насекомые» (*Insects Abroad*).

Глава 1

Введение

Биткойн – это совокупность концепций и технологий, которые составляют основу экосистемы цифровых денег. Единицы валюты, называемые биткойн, используются для хранения и передачи стоимости между участниками сети Биткойн. Пользователи этой сети общаются друг с другом посредством биткойн-протокола, в основном через интернет, хотя можно использовать и другие информационные сети. Стек протоколов Биткойна, доступный в виде программного обеспечения с открытым исходным кодом, может быть запущен на различных вычислительных устройствах, включая ноутбуки и смартфоны, что обеспечивает широкую доступность технологии.

В этой книге денежная единица называется «биткойн» (bitcoin) со строчной буквы «б», а система называется «Биткойн» (Bitcoin) с прописной буквы «Б».

Пользователи могут переводить биткойны по сети для выполнения практически всех операций, которые можно совершать с обычными валютами, включая покупку и продажу товаров, отправку денег частным лицам или организациям, а также выдачу кредитов. Биткойн можно покупать, продавать и обменивать на другие валюты на специализированных валютных биржах. Биткойн можно смело назвать идеальной формой интернет-денег, поскольку эта валюта работает быстро, безопасно и не знает границ.

В отличие от традиционных денег, биткойн полностью виртуален. Не существует ни физических монет, ни даже их цифровых версий. Монеты подразумеваются при совершении транзакций, в ходе которых стоимость передается от отправителя к получателю. Пользователи Биткойна контролируют ключи, с помощью которых можно подтвердить право собственности на биткойн в сети Биткойн. С помощью этих ключей можно подписывать транзакции с целью разблокировать их стоимость и потратить путем передачи новому владельцу. Ключи часто хранятся в цифровом кошельке на любом пользовательском компьютере или смартфоне.

Единственным условием для проведения операций с биткойнами является владение ключом, который дает возможность подписывать транзакции, – именно это передает полный контроль каждому пользователю.

Биткойн – это распределенная одноранговая (peer-to-peer) система. У нее отсутствует центральный сервер или центр управления. Единицы биткойна

получаются в результате процесса под названием «майнинг» (англ. mining – буквально «добыча»), который подразумевает многократное выполнение вычислительных задач, связанных со списком последних транзакций в сети Биткойн. Любой участник сети Биткойн может выступать в роли майнера, используя свои вычислительные устройства для безопасного проведения транзакций. Каждый майнер Биткойна в среднем за 10 минут может повысить безопасность прошедших транзакций, а в качестве вознаграждения он получает как совершенно новые биткойны, так и комиссионные за прошедшие транзакции. В сущности, майнинг биткойнов обеспечивает децентрализацию функций центрального банка по выпуску валюты и клирингу, заменяя собой необходимость в существовании какого-либо центрального банка.

Протокол Биткойна включает в себя встроенные алгоритмы для регулирования функции майнинга в сети. Сложность вычислительной задачи, которую должны выполнить майнеры, регулируется динамически, так что в среднем каждые 10 минут кто-то добивается успеха, независимо от того, сколько майнеров (и какой объем обработки) конкурируют в каждый момент времени. Протокол также постепенно уменьшает количество создаваемых биткойнов, что ограничивает общее количество биткойнов, которое когда-либо будет создано, фиксированной суммой чуть менее 21 млн монет. В результате количество биткойнов в обращении следует легкопредсказуемой кривой, по которой каждые четыре года в обращение поступает половина оставшихся монет. К моменту выпуска примерно 1 411 200 блоков, который предположительно произойдет в 2035 году, будет выпущено 99 % всех когда-либо существовавших биткойнов. Вследствие сокращения темпов эмиссии биткойна в долгосрочной перспективе эта валюта имеет дефляционный характер. Кроме того, никто не заставит вас принять биткойны, выпущенные сверх ожидаемого уровня эмиссии.

Помимо этого, Биткойн – это еще и название протокола, одноранговой сети и инновационной технологии распределенных вычислений. Биткойн опирается на десятилетия исследований в области криптографии и распределенных систем. Он включает в себя как минимум четыре ключевых новшества, собранных воедино в уникальную и мощную комбинацию. Биткойн – это:

- децентрализованная одноранговая (пиринговая) сеть (протокол Биткойн);
- общедоступный реестр транзакций (блокчейн);
- набор правил для независимой проверки (валидации) транзакций и эмиссии денежных средств (правила консенсуса);
- механизм достижения глобального децентрализованного консенсуса на действующем блокчейне с алгоритмом доказательства выполнения работы (Proof-of-Work).

С точки зрения разработчика, Биткойн можно рассматривать как подобие интернета денег, сеть для распространения ценностей и защиты прав собственности на цифровые активы с помощью распределенных вычислений. Биткойн – это нечто намного большее, чем может показаться на первый взгляд.

В этой главе речь идет о начале работы с объяснением некоторых основных концепций и терминов, о получении необходимого программного обеспечения и об использовании Биткойна для простых транзакций. В следующих

главах начнется изучение технологических уровней, обеспечивающих работу, а также внутренних механизмов сети и протокола Биткойна.

Цифровые валюты до Биткойна

Появление жизнеспособных цифровых денег тесно связано с развитием криптографии. Это неудивительно с учетом фундаментальных проблем при использовании битов для представления стоимости, которую можно обменять на товары и услуги. Три основных вопроса, с которыми сталкиваются пользователи цифровых денег:

- можно ли доверять подлинности денег, не являются ли они фальшивыми?
- можно ли быть уверенным в возможности потратить цифровые деньги только один раз (проблема, известная как doublespend – «двойная трата»)?
- можно ли быть уверенным, что больше никто, кроме меня, не сможет предъявить права на эти деньги?

Эмитенты бумажных денег постоянно решают проблему фальшивок, применяя все более сложные виды бумаги и технологии печати. Физические деньги легко решают проблему двойной траты, поскольку одна и та же бумажная купюра не может находиться в двух местах одновременно. Безусловно, обычные деньги также часто хранятся и передаются в цифровом виде. В этих случаях проблемы подделки и двойной траты решаются с помощью клиринга всех электронных транзакций через центральные структуры, которым известна вся валюта, находящаяся в обращении. Для цифровых денег, не имеющих аналогов в виде специальных чернил или голографических полос, основанием для доверия к легитимности заявления пользователя о ценности служит криптография. В частности, криптографические цифровые подписи дают пользователю возможность подписать цифровой актив или транзакцию, подтверждая его право собственности на него. При соответствующей архитектуре цифровые подписи также можно использовать для решения проблемы двойной траты.

По мере того как в конце 1980-х годов криптография становилась все более доступной и понятной, многие специалисты пытались использовать ее для создания цифровых денег. В рамках этих ранних проектов выпускались цифровые валюты, обычно обеспеченные национальной денежной системой или драгоценным металлом, например золотом.

Несмотря на работоспособность этих ранних цифровых валют, они имели централизованный характер и, как следствие, легко подвергались внешним атакам со стороны правительств и хакеров. Подобно традиционной банковской системе, ранние цифровые валюты использовали центральный расчетный аппарат для урегулирования всех транзакций через определенные промежутки времени. Некоторые из них потерпели впечатляющий крах при внезапной ликвидации материнской компании. Для защиты от вмешательства недоброжелателей, будь то законные правительства или преступные сообщества, нужна децентрализованная цифровая валюта, исключающая возможность единой точечной атаки. Биткойн – именно такая система, децентрализованная по своей сути и не имеющая никаких центральных органов власти или точек контроля, которые могли бы быть атакованы или дискредитированы.

История Биткойна

Впервые концепция Биткойна была описана в 2008 году в статье под названием «Биткойн: пиринговая система электронных денег» («Bitcoin: A Peer-to-Peer Electronic Cash System»)[1]. Материал был подписан псевдонимом Сатоши Накамото (Satoshi Nakamoto) (см. приложение А). В своей работе Накамото объединил несколько ранее сделанных изобретений, таких как цифровые подписи и система Hashcash, для создания полностью децентрализованной системы электронных денег. Эта система не нуждается в централизованном управлении для эмиссии валюты, расчетов и подтверждения транзакций. Ключевым новшеством стало использование распределенной системы вычислений – алгоритма «proof-of-work» («доказательство работы») – для проведения глобальной лотереи в среднем каждые 10 минут, что дает децентрализованной сети возможность прийти к консенсусу относительно состояния транзакций. Тем самым достигается эффективное решение проблемы двойной траты, при которой одна валютная единица может быть потрачена дважды. Ранее эта проблема считалась слабым местом цифровых валют и устранялась за счет клиринга всех транзакций через центральный расчетный центр.

Сеть Биткойн была запущена в 2009 году на основе эталонной разработки, созданной Накамото и впоследствии доработанной многими другими программистами. Количество и мощность систем, выполняющих алгоритм доказательства работы (proof-of-work, то есть майнинг) для обеспечения безопасности и устойчивости сети Биткойн, росли в геометрической прогрессии. В настоящее время их совокупная вычислительная мощность превышает общее количество вычислительных операций лучших суперкомпьютеров мира.

В апреле 2011 года Сатоши Накамото исчез из публичного пространства, возложив ответственность за разработку кода и сети на быстро растущую группу добровольных помощников. Кто именно создал биткойн, до сих пор остается невыясненным. Однако ни Сатоши Накамото, ни кто-либо другой не имеет единоличного контроля над системой Биткойн, поскольку ее работа основана на полностью прозрачных математических правилах, открытом исходном коде и общем согласии между участниками. Это изобретение само по себе является революционным и уже породило новую научно-техническую отрасль в области распределенных вычислений, экономики и эконометрики.

Решение проблемы распределенных вычислений

Изобретение Сатоши Накамото также является практически новым решением проблемы распределенных вычислений, известной как «проблема (задача) византийских генералов» (Byzantine Generals' Problem). Суть проблемы заключается в попытке договориться между несколькими субъектами без лидера о порядке действий путем обмена информацией по ненадежной и потенциально взломанной сети. Решение Сатоши Накамото, использующее концепцию доказательства работы для получения консенсуса *без привлечения центральной доверенной инстанции*, стало прорывом в области распределенных вычислений.

[1] «Bitcoin: A Peer-to-Peer Electronic Cash System», Satoshi Nakamoto.

Приступая к работе

Биткойн – это протокол, доступ к которому можно получить при помощи приложения с поддержкой этого протокола. По аналогии с веб-браузером, который является наиболее распространенным пользовательским интерфейсом для протокола HTTP, «биткойн-кошелек» (Bitcoin wallet) является наиболее распространенным пользовательским интерфейсом для системы Биткойн. Как и в примере со множеством веб-браузеров (например, Chrome, Safari и Firefox), существует множество реализаций и брендов биткойн-кошельков. И точно так же, как мы выбираем свои любимые браузеры, биткойн-кошельки могут отличаться по качеству, производительности, безопасности, конфиденциальности и надежности. Кроме того, имеется эталонная реализация протокола Биткойн, известная как «Bitcoin Core». Она создана на основе исходной концепции, разработанной Сатоши Накамото.

Выбор биткойн-кошелька

Биткойн-кошельки входят в число наиболее активно разрабатываемых приложений в экосистеме Биткойн. Здесь царит высокая конкуренция, и даже если прямо сейчас разрабатывается новый кошелек, несколько прошлогодних могут уже не поддерживаться. Выбор кошелька в значительной степени основан на личных предпочтениях, он определяется сферой применения и опытом пользователя. Поэтому нет смысла рекомендовать какой-либо определенный бренд или кошелек. Тем не менее можно классифицировать биткойн-кошельки в соответствии с их платформой и функциями, чтобы внести некоторую ясность относительно всех существующих типов кошельков. Стоит попробовать несколько разных вариантов, пока не найдется подходящий под конкретные нужды.

Разновидности биткойн-кошельков

Биткойн-кошельки в зависимости от платформы можно условно разделить на следующие категории.

Кошелек для настольных компьютеров
: Кошелек для ПК стал первым типом биткойн-кошелька, созданным в качестве эталонного варианта. Многие пользователи работают с кошельками для ПК по причине предлагаемых ими возможностей, автономности и управляемости. Однако запуск на распространенных операционных системах, таких как Windows и macOS, связан с определенными недостатками в плане обеспечения безопасности, поскольку эти платформы часто оказываются незащищенными и плохо настроенными.

Мобильный кошелек
: Мобильный кошелек можно назвать наиболее распространенным типом биткойн-кошелька. Эти приложения работают на операционных системах смартфонов, таких как Apple iOS и Android, и зачастую представляют собой

оптимальный выбор для начинающих. Многие из этих кошельков рассчитаны на простоту и удобство использования, но есть и полнофункциональные мобильные кошельки для опытных пользователей. Чтобы не загружать и не хранить большие объемы данных, многие мобильные кошельки получают информацию с удаленных серверов. Это снижает уровень конфиденциальности, поскольку позволяет третьим лицам получать сведения о ваших адресах и балансах биткойнов.

Веб-кошелек
: Доступ к веб-кошелькам открывается через веб-браузер, а сам кошелек находится на сервере стороннего оператора. Это похоже на веб-почту, поскольку полностью зависит от использования стороннего сервера. Некоторые из этих сервисов используют клиентский код, запускаемый в браузере пользователя, что дает возможность сохранить контроль над ключами Биткойна в руках владельца, хотя его зависимость от сервера все равно ставит конфиденциальность под угрозу. Однако большинство из подобных систем лишают пользователей контроля над ключами Биткойна в обмен на простоту использования. Хранить большие суммы биткойнов в сторонних системах нежелательно.

Аппаратные устройства подписи
: Аппаратные кошельки представляют собой оборудование для хранения ключей и подписывания транзакций с помощью специализированного аппаратного и микропрограммного обеспечения. Обычно они подключаются к настольному, мобильному или веб-кошельку через USB-кабель, беспроводной интерфейс ближнего поля (NFC) или камеру для работы с QR-кодами. Поскольку все операции с биткойном выполняются на специализированном оборудовании, такие кошельки менее уязвимы для различных типов атак. Иногда аппаратные устройства подписи называют «аппаратными кошельками», но для отправки и получения транзакций их нужно использовать в связке с полнофункциональным кошельком. Безопасность и конфиденциальность, обеспечиваемые этим кошельком в связке, играют решающую роль при использовании аппаратного устройства подписи.

Полнофункциональная или облегченная версия

Биткойн-кошельки можно также классифицировать по степени их автономности и способу взаимодействия с сетью Биткойн:

Полноценный узел (клиент)
: Полноценный узел (клиент) – это программа для проверки всей истории транзакций биткойна (каждая транзакция каждого пользователя, за все время). Опционально полноценные клиенты могут также хранить подтвержденные ранее транзакции и предоставлять данные другим программам Биткойна как на этом же компьютере, так и через интернет. Полноценный клиент использует значительные компьютерные ресурсы – примерно столько же, сколько часовой просмотр потокового видео за каждый день транзакций Биткойна, но при этом он обеспечивает пользователям полную автономность.

Облегченный клиент

Облегченный клиент, или клиент с упрощенной проверкой платежей (Simplified-Payment-Verification, SPV), подключается к полноценному клиенту либо другому удаленному серверу для получения и отправки информации о транзакциях биткойна, но в то же время обеспечивает локальное хранение пользовательского кошелька, частичную проверку получаемых транзакций и независимое создание исходящих транзакций.

Сторонний API-клиент

Сторонний API-клиент предназначен для взаимодействия с сетью Биткойн через систему API сторонних разработчиков, а не путем прямого подключения. Кошелек может храниться у пользователя или на сторонних серверах, но при этом клиент полностью доверится удаленному серверу, который будет передавать ему точную информацию и защищать его конфиденциальность.

Биткойн является одноранговой (пиринговой – peer-to-peer, P2P) сетью. Полноценные узлы (клиенты) сети – это равноправные пользователи (пиры): каждый из них индивидуально проверяет достоверность всех подтвержденных транзакций и может предоставлять данные своему пользователю с максимальной степенью достоверности. Облегченные кошельки и другое программное обеспечение являются *клиентами*: каждый клиент зависит от одного или нескольких пиров, которые предоставляют ему достоверные данные. Биткойн-клиенты могут выполнять вторичную проверку некоторой части получаемых ими данных и устанавливать соединения с несколькими пирами для уменьшения зависимости от подлинности отдельного пира, но в конечном итоге безопасность клиента зависит от надежности его пиров.

Кто контролирует ключи

Очень важным аспектом является вопрос о том, *кто контролирует ключи*. Как будет показано в последующих главах, доступ к биткойнам контролируется «секретными ключами» (private keys), которые похожи на очень длинные PIN-коды. Если вы единственный, кто имеет право распоряжаться этими секретными ключами, значит, вы контролируете свои биткойны. И наоборот, если контроль над ними вам не принадлежит, вашими биткойнами управляет третье лицо, которое в конечном итоге контролирует ваши средства от вашего имени. Программное обеспечение для управления ключами разделяется на две важные категории по принципу контроля: *кошельки*, где ключи и денежные средства вы контролируете, и счета в хранилищах, где ключами управляет третья сторона. Чтобы особо отметить этот момент, один из авторов (Андреас) придумал такую фразу: *Твои ключи – твои монеты* (Your keys, your coins). *Не твои ключи – не твои монеты*.

Если совместить все эти классификации, можно выделить несколько основных типов биткойн-кошельков, среди которых наиболее распространены три: полнофункциональные кошельки (сетевые узлы) для настольных компьютеров (вы контролируете ключи), облегченные мобильные кошельки (вы контролируете ключи) и веб-аккаунты сторонних разработчиков (вы не контролируете ключи). Границы между различными категориями иногда размыты, поскольку программы работают на разных платформах и могут по-разному взаимодействовать с сетью.

Быстрый старт

Условная Алиса – пользователь, далекий от техники, и только недавно узнала о Биткойне от своего друга Джо. На вечеринке Джо восторженно рассказал всем собравшимся о Биткойне и устроил демонстрацию. Заинтригованная Алиса поинтересовалась, как можно начать работать с Биткойном. Джо сказал, что для новичков лучше всего подойдет мобильный кошелек, и порекомендовал несколько своих любимых вариантов. Алиса скачала один из рекомендованных Джо кошельков и установила его на свой телефон.

Когда Алиса впервые запустила приложение кошелька, она выбрала опцию создания нового биткойн-кошелька. Поскольку выбранный ею кошелек является некастодиальным (noncustodial), то есть не управляется извне, Алиса (и только Алиса) будет контролировать свои ключи. Поэтому она несет ответственность за их резервное копирование, ибо потеря ключей означает потерю доступа к биткойнам. Чтобы облегчить эту задачу, ее кошелек генерирует *код восстановления* (*recovery code*), который можно использовать для восстановления кошелька.

Коды восстановления

Большинство современных некастодиальных биткойн-кошельков предоставляют пользователю код восстановления для резервного копирования. Обычно этот код состоит из цифр, букв или слов, выбранных программным обеспечением случайным образом. Он используется в качестве основы для генерации кошельком новых ключей. Примеры см. в табл. 1.1.

Таблица 1.1. Примеры кодов восстановления

Кошелек	Код восстановления
BlueWallet	(1) media (2) suspect (3) effort (4) dish (5) album (6) shaft (7) price (8) junk (9) pizza (10) situate (11) oyster (12) rib
Electrum	nephew dog crane clever quantum crazy purse traffic repeat fruit old clutch
Muun	LAFV TZUN V27E NU4D WPF4 BRJ4 ELLP BNFL

Код восстановления иногда называют «мнемоникой» (mnemonic), или «мнемонической фразой» (mnemonic phrase), что подразумевает, что вы должны запомнить эту фразу, ибо запись фразы на бумаге требует меньше усилий и, как правило, более надежна, чем память большинства людей. Другое альтернативное название – «семенная фраза» (seed phrase), потому что она обеспечивает всходы (из «семени» – seed) для функции, которая генерирует все ключи кошелька.

Если с кошельком Алисы что-то случится, она сможет загрузить новую копию программы и ввести код восстановления для повторного создания базы данных кошелька со всеми транзакциями, которые она когда-либо отправляла или получала. Однако сам по себе код восстановления не поможет вернуть дополнительные данные, которые Алиса ввела в свой кошелек, например метки, связанные с определенными адресами или транзакциями. Хотя потеря доступа к этим метаданным не так важна, как потеря доступа к деньгам, она все же может быть по-своему значимой. Представьте, что вам нужно просмотреть старую выписку по банковской или кредитной карте, а имя каждого субъек-

та, которому вы заплатили (или который заплатил вам), было удалено. Чтобы предотвратить потерю метаданных, многие кошельки, помимо кодов восстановления, имеют дополнительную функцию резервного копирования.

Сегодня для некоторых кошельков эта дополнительная функция резервного копирования даже важнее, чем раньше. Многие биткойн-платежи теперь осуществляются *вне цепочки блокчейна* (технология *offchain*), когда не все платежи хранятся в открытом блокчейне. Это снижает затраты пользователя, повышает конфиденциальность и дает ряд других преимуществ. Однако это означает, что механизмы, подобные кодам восстановления, которые зависят от данных в цепочке (onchain), не могут гарантировать восстановление всех биткойнов пользователя. Для приложений с поддержкой технологии offchain важно регулярно создавать резервные копии базы данных кошелька.

Обратите внимание, что при первом получении средств на новый мобильный кошелек многие из них часто проверяют, надежно ли сохранен код восстановления. Это может включать в себя как простой запрос, так и требование вручную ввести повторный код.

Многие легальные кошельки запрашивают повторный ввод кода восстановления, однако существуют и вредоносные программы, которые имитируют дизайн кошелька, заставляют вас ввести код восстановления, а затем передают его разработчику вредоносной программы, чтобы он мог украсть ваши средства. Это похоже на фишинговые веб-сайты, которые пытаются обманом заставить вас сообщить им банковский пароль. Большинство приложений для кошельков запрашивают код восстановления только при первоначальной настройке (до получения биткойнов) и при восстановлении (после потери доступа к исходному кошельку). Если приложение запрашивает код восстановления в любое другое время, проконсультируйтесь с экспертом для гарантии защиты от мошенников.

Биткойн-адреса

Теперь Алиса готова использовать свой новый биткойн-кошелек. Приложение ее кошелька в случайном порядке генерирует секретный ключ (более подробно описано в разделе «Секретные ключи»). Он будет использоваться для получения биткойн-адресов, направляемых в ее кошелек. На данном этапе ее биткойн-адреса еще неизвестны сети Биткойн и не «зарегистрированы» в какой-либо части системы Биткойн. Ее биткойн-адреса – это просто числа, соответствующие ее секретному ключу, который она может использовать для контроля доступа к финансам. Адреса генерируются ее кошельком автономно, без привязки или регистрации в каком-либо сервисе.

Существует множество адресов и форматов счетов Биткойна. Этими адресами и счетами можно делиться с другими пользователями системы, которые могут использовать их для отправки биткойнов непосредственно на ваш кошелек. Адрес или счет можно передавать другим людям без опасений за сохранность ваших биткойнов. В отличие от номера банковского счета, ни один человек, узнавший один из ваших биткойн-адресов, не сможет снять деньги с вашего кошелька – инициатором всех операций должны быть вы. Однако если вы дадите двум людям один и тот же адрес, они смогут узнать, сколько биткойнов отправил вам другой пользователь. Если вы опубликуете свой адрес в открытом доступе, все смогут узнать, сколько биткойнов отправили на этот адрес другие люди. Чтобы защитить свою конфиденциальность, каждый раз при запросе платежа необходимо генерировать новый счет с новым адресом.

Получение биткойнов

Алиса нажимает кнопку «Получить» на экране получения средств (Receive) своего мобильного биткойн-кошелька, после чего появляется QR-код, как на рис. 1.1.

Рис. 1.1 ❖ Экран адреса биткойн-кошелька в формате QR-кода

QR-код – это квадрат с узором из черных и белых точек, представляющий собой разновидность штрих-кода с той же информацией, но в формате, который может быть отсканирован камерой смартфона Джо.

Любые средства, отправленные на адреса из этой книги, будут потеряны. Если вы хотите протестировать отправку биткойнов, пожалуйста, рассмотрите вариант пожертвования в благотворительную организацию, принимающую биткойны.

Получение вашего первого биткойна

Первое, что необходимо сделать новым пользователям, – приобрести немного биткойнов.

Транзакции с биткойнами необратимы. Большинство электронных платежных систем, таких как кредитные и дебетовые карты, PayPal и переводы на банковские счета, являются обратимыми. Для продавца биткойнов эта особен-

ность означает очень высокий риск того, что покупатель отменит электронный платеж после получения биткойнов, фактически обманув продавца. Для снижения этого риска в компаниях, принимающих традиционные электронные платежи в обмен на биткойны, обычно требуют от покупателей пройти проверку личности и кредитоспособности, что может занять несколько дней или недель. Для нового пользователя это может означать отсутствие возможности мгновенной покупки биткойнов с помощью кредитной карты. Тем не менее если проявить немного терпения и творческого подхода, в этом не будет необходимости.

Вот несколько способов покупки биткойнов для новых пользователей:

- найдите знакомого с биткойнами и купите у него или у нее напрямую. Многие начинающие биткойн-пользователи поступают именно так. Такой способ является наименее сложным. Одна из возможностей познакомиться с обладателями биткойнов – посещение местных биткойн-встреч, указанных на сайте Meetup.com;
- зарабатывайте биткойны, продавая товар или услугу за биткойны. Например, программист может продать свои навыки программирования. Если вы парикмахер, стригите волосы за биткойны;
- воспользуйтесь биткойн-банкоматом в вашем городе. Такой банкомат принимает наличные и отправляет биткойны на биткойн-кошелек вашего смартфона;
- используйте валютную биржу биткойнов, привязанную к вашему банковскому счету. Сегодня во многих странах существуют валютные биржи, на которых покупатели и продавцы могут обменивать биткойны на местную валюту. Сервисы по обмену валют, такие как BitcoinAverage, часто показывают список бирж биткойнов для каждой валюты.

Одним из преимуществ Биткойна перед другими платежными системами является повышенная конфиденциальность при правильном его использовании. Приобретение, хранение и траты биткойнов не требуют разглашения конфиденциальных и личных данных третьим лицам. Однако при контакте Биткойна с традиционными системами, такими как валютные биржи, зачастую вступают в силу национальные и международные правила. Чтобы обменять биткойн на национальную валюту, часто требуется предоставить документы, удостоверяющие личность, и банковскую информацию. Пользователям следует помнить, что после привязки адреса Биткойна к личности другие связанные с ним транзакции также могут быть легко идентифицированы и отслежены, включая совершенные ранее. Это одна из причин, по которой многие пользователи предпочитают иметь специальные обменные счета, не привязанные к их кошелькам.

Алису познакомил с Биткойном ее друг, так что она без особых затруднений приобрела свои первые биткойны. Далее речь пойдет о покупке биткойнов у ее друга Джо и о том, как Джо отправляет биткойны на ее кошелек.

Определение актуальной цены биткойна

Перед тем как Алиса сможет купить биткойны у Джо, они должны договориться об обменном курсе биткойна к доллару США. В связи с этим у новичков в мире

биткойна возникает типичный вопрос: «Кто устанавливает цену на биткойны»? Короткий ответ таков: цену устанавливает рынок.

Биткойн, как и большинство других валют, имеет плавающий обменный курс. Это означает, что его стоимость колеблется в зависимости от спроса и предложения на тех или иных рынках, где он торгуется. Например, «цена» биткойна в долларах США рассчитывается на каждом рынке на основе последних торгов биткойнами и долларами США. Таким образом, курс имеет тенденцию к колебаниям с периодичностью несколько раз в секунду. Сервис ценообразования агрегирует цены с нескольких рынков и рассчитывает средневзвешенную по объему цену, отражающую общий рыночный обменный курс валютной пары (например, BTC/USD, биткойн/доллар США).

Существуют сотни приложений и веб-сайтов, которые могут предоставить текущий рыночный курс. Вот некоторые из наиболее популярных:

Bitcoin Average (https://bitcoinaverage.com/)
: Сайт предоставляет возможность простого просмотра средневзвешенного по объему показателя для каждой валюты.

CoinCap (https://coincap.io/)
: Сервис предоставляет список рыночной капитализации и обменных курсов сотен криптовалют, включая биткойны.

Chicago Mercantile Exchange (CME) Bitcoin Reference Rate (https://www.cmegroup.com/markets/cryptocurrencies/cme-cf-cryptocurrency-benchmarks.html)
: Эталонный курс, который может быть использован для институциональных и контрактных расчетов. Предоставляется CME в качестве компонента данных для инвестиций.

Помимо этих сайтов и приложений, некоторые биткойн-кошельки позволяют автоматически конвертировать суммы в биткойнах и других валютах.

Отправка и получение биткойнов

Алиса решила купить 0,001 биткойна. После того как они с Джо проверили обменный курс, Алиса отдает Джо соответствующую сумму наличных, открывает приложение мобильного кошелька и выбирает пункт «Получить». На экране появляется QR-код с первым биткойн-адресом Алисы.

Затем Джо выбирает в кошельке своего смартфона пункт «Отправить» и запускает сканер QR-кодов. Это позволяет ему отсканировать код с помощью камеры смартфона, чтобы не вводить достаточно длинный Bitcoin-адрес Алисы.

Джо указал биткойн-адрес Алисы в качестве получателя. Он вводит сумму в размере 0,001 биткойна (BTC), см. рис. 1.2. Некоторые кошельки могут показывать сумму в другом номинале: 0,001 BTC равен 1 миллибиткойну (mBTC), или 100 000 сатоши (sats).

Некоторые кошельки могут также предложить Джо ввести метку для этой транзакции. В этом случае Джо вводит «Алиса». Через несколько недель или месяцев это поможет Джо вспомнить причину отправки этих 0,001 биткойна. Некоторые кошельки также могут подсказать Джо о комиссиях. В зависимости

от типа кошелька и способа отправки транзакции приложение кошелька может попросить Джо либо ввести размер комиссии за транзакцию, либо указать предлагаемую комиссию (или ставку комиссии). Чем выше комиссия за транзакцию, тем быстрее она будет подтверждена (см. раздел «Подтверждения»).

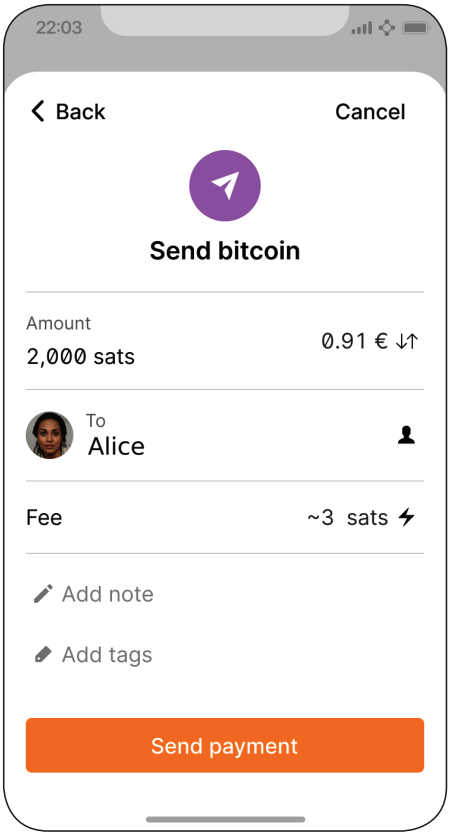


Рис. 1.2 ❖ Экран отправки биткойн-кошелька

После этого Джо тщательно проверяет корректность введенной суммы, поскольку скоро ему предстоит перевести деньги и допущенные ошибки станут необратимыми. Перепроверив адрес и сумму, он нажимает «Отправить», чтобы совершить транзакцию. Мобильный биткойн-кошелек Джо формирует транзакцию для перевода 0,001 BTC на адрес, указанный Алисой, берет средства из кошелька Джо и подписывает транзакцию его секретными ключами. Таким образом, сеть Биткойн получает информацию о подтверждении перевода средств на новый адрес Алисы. Поскольку транзакция передается по пиринговому протоколу, она стремительно распространяется по сети Биткойн. Уже через несколько секунд большинство подключенных к сети узлов получат данные о транзакции и впервые узнают адрес Алисы.

Тем временем кошелек Алисы постоянно «слушает» новые транзакции в сети Биткойн, выявляя совпадения с содержащимися в нем адресами. Через

несколько секунд после передачи транзакции из кошелька Джо в кошелек Алисы придет сообщение о том, что она получила 0,001 BTC.

Подтверждения

Поначалу транзакция от Джо будет отображаться в адресах Алисы как «Неподтвержденная» (Unconfirmed). Это означает, что операция распространилась по сети, но еще не записана в журнал транзакций Биткойна – так называемый блокчейн. Для подтверждения транзакция должна быть включена в блок и добавлена в блокчейн, что происходит в среднем каждые 10 минут. В традиционных финансовых терминах это принято называть клирингом (clearing). Подробнее о распространении, проверке и клиринге (подтверждении) транзакций биткойна читайте в главе 12.

Теперь Алиса стала гордой обладательницей 0,001 BTC, которые она может потратить. В следующие несколько дней Алиса покупает больше биткойнов через банкомат и биржу. В следующей главе на примере ее первой покупки в биткойнах будут более подробно рассмотрены базовые технологии транзакций и распространения.

Глава 2

Как устроен Биткойн

В отличие от традиционных банковских и платежных систем, система Биткойн не требует доверительного управления сторонними структурами. Вместо единого центрального доверенного органа каждый пользователь может проверить корректность работы каждого элемента системы Биткойн с помощью программного обеспечения на своем компьютере. В этой главе приводится общее описание биткойна: на примере одной транзакции, проходящей через систему Биткойн, будет показано, как она записывается в блокчейн – распределенный журнал всех транзакций. В последующих главах будут рассмотрены технологии, лежащие в основе транзакций, сети и майнинга.

Общие сведения о Биткойне

Система Биткойн включает пользователей с кошельками для ключей и транзакций, которые распространяются по сети, а также майнеров, создающих (путем конкурирующих вычислений) блокчейн консенсуса, который представляет собой достоверный журнал всех транзакций.

Все примеры в этой главе основаны на реальных транзакциях в сети Биткойн и представляют примеры взаимодействия нескольких пользователей при отправке средств с одного кошелька на другой. Прослеживая транзакцию через сеть Биткойн к блокчейну, можно использовать сайт *проводника блокчейна* (*blockchain explorer*) для визуализации каждого шага. Проводник блокчейна (или просто «проводник блока» – block explorer) представляет собой веб-приложение, которое работает как поисковая система Биткойн, позволяющая находить адреса, транзакции и блоки, а также просматривать взаимосвязи и потоки между ними.

К числу наиболее популярных проводников блокчейна можно отнести следующие:

- Blockstream Explorer (https://blockstream.info/);
- Mempool.Space (https://mempool.space/);
- BlockCypher Explorer (https://live.blockcypher.com/).

Каждый из них имеет функцию поиска, которая способна по адресу Биткойна, хешу транзакции, номеру блока или хешу блока извлечь соответствующую информацию из сети Биткойн. Для каждого примера транзакции или блока указан URL, чтобы можно было самостоятельно найти его и подробно изучить.

Предупреждение о конфиденциальности проводника блокчейна

Поиск информации в блокчейн-проводнике может рассказать его оператору о вашем интересе к этой информации. Это позволит ему связать ее с вашим IP-адресом, данными браузера, прошлыми поисками или другой легко определяемой информацией. Если просмотреть транзакции в этой книге, оператор блокчейн-проводника может догадаться, что человек изучает Биткойн, и это не должно быть проблемой. Но если просмотреть собственные транзакции, оператор может узнать, сколько биткойнов было получено, потрачено и чем вы владеете на данный момент.

Покупка в интернет-магазине

Представленная в предыдущей главе Алиса – новый пользователь, который только что обзавелся своими первыми биткойнами. В разделе «Получение первого биткойна» Алиса встретилась со своим другом Джо, чтобы обменять немного наличных на биткойны. С тех пор Алиса приобрела еще несколько биткойнов. Теперь Алиса совершит свою первую транзакцию. Она купит доступ к премиальному эпизоду подкаста в интернет-магазине Боба.

Интернет-магазин Боба не так давно начал принимать платежи в биткойнах, добавив на свой сайт опцию «Биткойн». Цены в магазине Боба указаны в местной валюте (долларах США), но при оформлении заказа покупатели могут заплатить либо в долларах, либо в биткойнах.

Алиса находит эпизод подкаста, который она хочет купить, и переходит на страницу оформления заказа. В процессе оформления в дополнение к обычным вариантам Алисе предлагается оплатить покупку биткойнами. В корзине отображается цена в долларах США, а также в биткойнах (BTC) по актуальному курсу биткойна.

Система электронной коммерции Боба автоматически создает QR-код, содержащий счет-фактуру (рис. 2.1).

Рис. 2.1 ❖ QR-код счета-фактуры

В отличие от QR-кода, который просто содержит адрес получателя биткойна, этот счет представляет собой закодированный в QR-коде URI с адресом получателя, суммой платежа и его описанием. Это позволяет приложению биткойн-кошелька заполнять информацию, используемую для отправки платежа, и при этом показывать понятное описание. Можно отсканировать QR-код с помощью приложения биткойн-кошелька и посмотреть, что увидит Алиса:

```
bitcoin:bc1qk2g6u8p4qm2s2lh3gts5cpt2mrv5skcuu7u3e4?amount=0.01577764&
label=Bob%27s%20Store&
message=Purchase%20at%20Bob%27s%20Store

Components of the URI

A Bitcoin address: "bc1qk2g6u8p4qm2s2lh3gts5cpt2mrv5skcuu7u3e4"
The payment amount: "0.01577764"
A label for the recipient address: "Bob's Store"
A description for the payment: "Purchase at Bob's Store"
```

Попробуйте отсканировать его своим кошельком, чтобы увидеть адрес и сумму, но НЕ ОТПРАВЛЯЙТЕ ДЕНЬГИ.

Алиса сканирует штрих-код на дисплее с помощью смартфона. Ее смартфон показывает платеж в магазин Боба на нужную сумму, и она выбирает «Отправить», чтобы авторизовать платеж. Через несколько секунд (примерно столько же, сколько занимает авторизация кредитной карты) Боб видит транзакцию на кассовом аппарате.

В сети Биткойн можно совершать операции с дробными величинами, начиная с миллибиткойна (1/1000 часть биткойна) и заканчивая 1/100 000 000 000 частью биткойна, которая известна как сатоши (satoshi). В этой книге для обозначения сумм более одного биткойна и при использовании десятичной системы счисления, например «10 биткойнов» или «0,001 биткойна», используются правила множественного числа, применяемые к долларам и другим традиционным валютам. Аналогичные правила применяются и к другим единицам учета биткойнов, таким как миллибиткойны и сатоши.

Для изучения данных блокчейна, таких как платеж, произведенный Бобу в транзакции Алисы, можно использовать проводник блоков.

В следующих разделах эта транзакция будет рассмотрена более подробно. Как кошелек Алисы создал ее, как она распространилась по сети, как была проверена и, наконец, как Боб может потратить эту сумму в последующих транзакциях.

Транзакции биткойна

Выражаясь простым языком, транзакция сообщает сети, что владелец некоторого количества биткойнов санкционировал их передачу другому пользователю. Новый владелец может потратить биткойны, создав еще одну транзакцию, которая авторизует передачу другому владельцу, и так далее, по цепочке владения.

Входные и выходные данные транзакций

Транзакции похожи на строки в бухгалтерской книге с двойной записью. Каждая транзакция содержит одну или несколько *входных позиций* (*inputs*) по расходуемым средствам. На другой стороне транзакции есть одна или несколько *выходных точек* (*outputs*), на которые поступают средства. Входные и выходные данные не обязательно складываются в одну и ту же сумму. Напротив, сумма на выходе немного меньше, чем на входе, а разница представляет собой предполагаемую *комиссию за транзакцию* (*transaction fee*). Она представляет собой небольшой платеж, который взимается майнером за внесение транзакции в блокчейн. Транзакция биткойна показана на рис. 2.2 в виде записи в бухгалтерской книге.

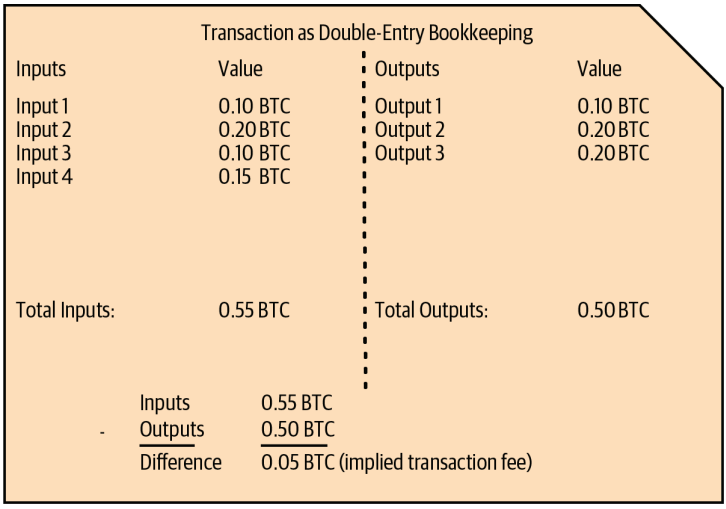


Рис. 2.2 ❖ Транзакция в виде бухгалтерской книги с двойной записью

Кроме того, транзакция также включает в себя доказательство права собственности на каждое количество расходуемых биткойнов в виде цифровой подписи владельца, которая может быть подтверждена независимым образом любыми лицами. В терминологии Биткойна расходование – это подпись транзакции, которая передает стоимость от предыдущей транзакции новому владельцу, идентифицированному по адресу Биткойна.

Цепочки транзакций

Платеж Алисы в магазин Боба предусматривает использование выходных данных предыдущей транзакции в качестве входных данных. В предыдущей главе Алиса получила биткойны от своего друга Джо в обмен на наличные. На рис. 2.3 это действие обозначено как «*Транзакция 1*» (Tx1).

С помощью Tx1 было отправлено 0,001 биткойна (100 000 сатоши) на выход, закрытый ключом Алисы. Ее новая транзакция в магазин Боба (Tx2) ссылается

на ранее использованный результат в качестве входных данных. На рисунке эта ссылка показана стрелкой, а вход обозначен как «Tx1:0». В реальных транзакциях эта ссылка представляет собой 32-байтовый идентификатор транзакции (txid) для той операции, в результате которой Алиса получила деньги от Джо. Символы «:0» указывают на позицию выхода, где Алиса получила деньги. В данном случае это первая позиция (позиция 0).

Как показано на рисунке, фактические транзакции Биткойна не содержат явного указания стоимости входа. Чтобы определить значение входных данных, программе требуется обратиться к ссылке на входные данные и найти предыдущие данные транзакции о потраченных средствах.

Транзакция Tx2 Алисы содержит два новых выхода, один из которых оплачивает 75 000 сатоши за подкаст, а другой возвращает 20 000 сатоши Алисе в качестве сдачи.

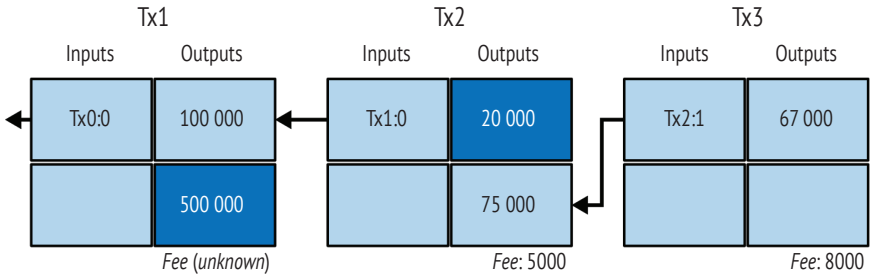


Рис. 2.3 ❖ Цепочка транзакций,
где выход одной транзакции является входом следующей

Последовательные транзакции Биткойна – это формат данных, который используется программами для отправки транзакций. Он кодирует передаваемое значение с помощью целого числа наименьшей определенной единицы стоимости в цепочке. Когда Биткойн только создавался, у этой единицы не было названия, и некоторые разработчики просто называли ее *базовой единицей* (*base unit*). Позже многие пользователи стали называть эту единицу *сатоши* (satoshi, sat) в честь создателя Биткойна. На рис. 2.3 и некоторых других иллюстрациях в этой книге используются значения сатоши, потому что именно они используются в самом протоколе.

Выдача сдачи

В дополнение к одному или нескольким выходам для выплаты получателю биткойнов во многих транзакциях также присутствует выход для выплаты тратившему биткойны, называемый *выходом сдачи* (*change output*). Так происходит потому, что денежные средства, такие как банкноты, невозможно потратить по частям. Если вы покупаете в магазине товар стоимостью $5, но используете для оплаты купюру в $20, то ожидаете получить $15 в качестве сдачи. Та же концепция применима и к транзакциям в Биткойне. Если бы вы купили товар стоимостью 5 биткойнов, но у вас был только вход стоимостью 20 биткойнов, вы бы отправили один выход в 5 биткойнов владельцу магазина и один выход в 15 биткойнов обратно себе в качестве сдачи (без учета комиссии за транзакцию).

На уровне протокола Биткойна нет никакой разницы между выходом сдачи (и адресом ее оплаты, называемым *адресом сдачи – change address*) и выходом платежа.

Важно отметить, что адрес сдачи не обязательно должен совпадать с адресом ввода, и часто, в целях конфиденциальности, он представляет собой новый адрес из кошелька владельца. В идеале эти два варианта применения выхода должны использовать неизвестные ранее адреса и в остальном выглядеть идентично, что не позволит третьей стороне определить, какие из них являются сдачей, а какие – платежами. Однако для наглядности на рис. 2.3 к выходам сдачи добавлено затенение.

Не каждая транзакция предусматривает выход сдачи. Не имеющие такового называют *неизменными транзакциями* (*changeless transactions*), и они могут иметь только один выход. Неизменные транзакции практичны лишь в том случае, если расходуемая сумма примерно равна имеющейся на входе транзакции за вычетом предполагаемой комиссии за ее проведение. На рис. 2.3 показано, как Боб создает Tx3 в качестве неизменной транзакции для расходования суммы, полученной им в Tx2.

Выбор номинала монет

Различные кошельки придерживаются разных стратегий при выборе исходных данных для совершения платежа. Это называют *выбором номинала монет* (*coin selection*).

Можно объединить несколько мелких входов или использовать единственный, равный желаемому платежу или превышающий его. Если кошелек не умеет объединять входы для точного соответствия желаемому платежу плюс комиссии за транзакцию, ему придется генерировать некоторое количество сдачи. Это очень похоже на обращение с наличностью. Если постоянно использовать самую крупную банкноту, то в итоге карман будет полон мелочи. Если же использовать только мелочь, то в итоге в кармане останутся только крупные купюры. Люди подсознательно находят баланс между этими крайностями, и разработчики биткойн-кошельков тоже стараются найти этот баланс.

Типичные формы транзакций

Весьма распространенной формой транзакции является *простой платеж* (*simple payment*). Этот тип транзакции с одним входом и двумя выходами показан на рис. 2.4.

Еще одной распространенной формой транзакции можно назвать *транзакцию консолидации* (*consolidation transaction*), в которой несколько входов преобразуются в один выход (рис. 2.5). В реальности это похоже на обмен кучки монет и мелких купюр на одну более крупную банкноту. Подобные транзакции иногда генерируются кошельками и фирмами для обработки множества мелких сумм.

И еще одна часто встречающаяся в блокчейне форма транзакций – *пакетные платежи* (*payment batching*), которые проводятся на несколько выходов для

нескольких получателей (рис. 2.6). Этот тип транзакций иногда используется коммерческими организациями для распределения средств, как, например, при выплате заработной платы нескольким сотрудникам.

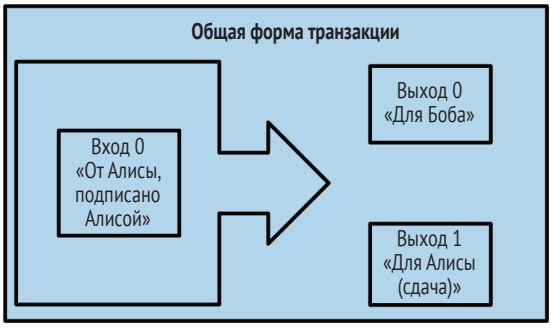


Рис. 2.4 ❖ Наиболее распространенная форма транзакции

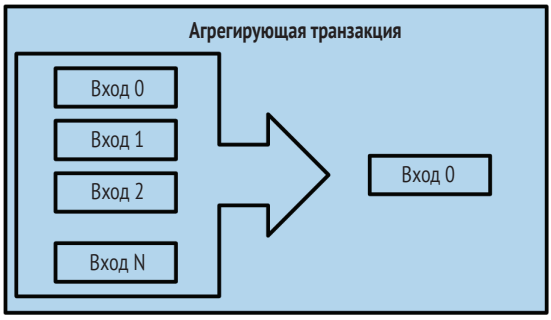


Рис. 2.5 ❖ Транзакция консолидации для объединения средств

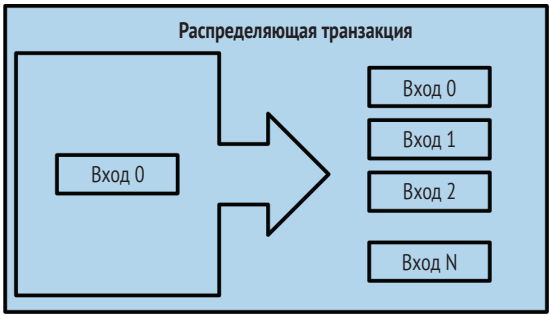


Рис. 2.6 ❖ Пакетная транзакция для распределения средств

Построение транзакции

Приложение кошелька Алисы содержит всю необходимую логику для выбора входов и генерации выходов при построении транзакции в соответствии с тре-

бованиями Алисы. Ей нужно указать лишь целевой адрес, сумму и комиссию за транзакцию, и все остальное происходит в приложении кошелька, без ее участия. Важно отметить, что когда в кошельке уже есть информация о контролируемых входах, он может создавать транзакции даже в условиях полного отсутствия связи. Подобно чеку, который можно выписать дома, а затем отправить в банк в конверте, транзакцию не обязательно создавать и подписывать при наличии подключения к сети Биткойн.

Получение правильных входных данных

Приложение кошелька Алисы сначала должно найти входы для оплаты суммы, которую она собирается отправить Бобу. Большинство кошельков отслеживают все возможные выходы, относящиеся к адресам в кошельке. Поэтому в кошельке Алисы будет храниться копия выхода транзакции Джо, созданной в обмен на наличные (см. раздел «Получение вашего первого биткойна»). Приложение биткойн-кошелька, которое работает на полноценном узле, фактически содержит копию всех подтвержденных транзакций, называемых *неизрасходованными выходами транзакций* (*Unspent Transaction Output*, *UTXO*). Поскольку полнофункциональные узлы используют большие ресурсы, многие пользовательские кошельки запускают облегченных клиентов, которые позволяют отслеживать только собственные UTXO пользователя.

В этом случае для оплаты подкаста достаточно одного UTXO. В ином случае приложению кошелька Алисы, возможно, пришлось бы объединить несколько более мелких UTXO, словно собирая монетки из кошелька, пока не удалось бы наскрести достаточно для оплаты подкаста. В обоих случаях может возникнуть необходимость получения сдачи, что будет рассмотрено в следующем разделе, когда приложение кошелька будет создавать выходные данные транзакции (платежи).

Создание выходов

Транзакционный выход создается с помощью скрипта, в котором написано что-то вроде: «Этот выход будет выплачен любому предъявителю подписи от ключа, соответствующего открытому адресу Боба». Поскольку только Боб владеет кошельком с соответствующими этому адресу ключами, только кошелек Боба может представить такую подпись для последующего расходования этих средств. Поэтому Алиса поместит в выходное значение требование подписи от Боба.

Эта транзакция также включает второй выход, потому что средства Алисы содержат больше денег, чем стоит подкаст. Выход сдачи для Алисы создается в той же транзакции, что и платеж Бобу. Фактически кошелек Алисы разбивает ее средства на два выхода: один для Боба и один обратно для нее самой. Она сможет потратить деньги со сдачи в последующей транзакции.

Наконец, для своевременной обработки транзакции сетью приложение кошелька Алисы добавит небольшую комиссию. Эта плата не указывается в транзакции в явном порядке, но подразумевается в результате разницы в стоимости

между входом и выходом. Эта плата за транзакцию взимается майнером как плата за включение транзакции в блок, который записывается в блокчейн.

По ссылке можно просмотреть транзакцию от Алисы в магазин Боба: https://blockstream.info/tx/466200308696215bbc949d5141a49a4138ecdfdfaa2a8029c1f9bcecd1f96177.

Добавление транзакции в блокчейн

Созданная в кошельке Алисы транзакция содержит все необходимое для подтверждения права собственности на средства и назначения новых владельцев. Теперь транзакция должна быть передана в сеть Биткойн, где она станет частью блокчейна. В следующем разделе будет показано, как транзакция становится частью нового блока и как происходит майнинг блока. Наконец, будет показано, как новый блок, добавленный в блокчейн, обретает все большее доверие в сети по мере добавления новых блоков.

Передача транзакции

Поскольку транзакция содержит всю необходимую для ее обработки информацию, не имеет значения, как и где она будет передана в сеть Биткойн. Эта сеть является пиринговой (одноранговой), где каждый пиринговый пользователь Биткойна принимает участие в ее работе, подключаясь к нескольким таким же пиринговым пользователям. Целью сети Биткойн является распространение транзакций и блоков среди всех участников.

Как распространяются транзакции и блоки

Пиры в одноранговой сети Биткойн – это приложения с программной логикой и необходимыми данными для полной проверки корректности новой транзакции. Связи между пирами часто изображают в виде граней (линий) на схеме, а сами пиры – в виде узлов (точек). По этой причине пиры Биткойна принято называть «узлами полной верификации» (full verification nodes), или просто «полными узлами» (full nodes).

Кошелек Алисы может отправить новую транзакцию любому узлу Биткойна посредством любого типа соединения: проводного, Wi-Fi, мобильного и т. д. Он также может отправить транзакцию другой программе (например, проводнику блокчейна), которая перенаправит ее на узел сети. При этом биткойн-кошелек Алисы не обязательно должен быть подключен к биткойн-кошельку Боба напрямую, и ей вовсе не обязательно использовать соединение с интернетом, предлагаемое Бобом, хотя оба этих варианта вполне допустимы. Если узел сети Биткойн получает корректную транзакцию, которую он раньше не встречал, он передает ее на все остальные узлы, к которым он подключен. Данный способ распространения известен как «*сплетни*» (*gossiping*). Благодаря этому транзакция быстро распространяется по пиринговой сети и достигает большого количества узлов за несколько секунд.

С точки зрения Боба

Если кошелек Боба напрямую соединен с кошельком Алисы, он может получить транзакцию первым. Однако и в том случае, если кошелек Алисы отправит транзакцию через другие узлы, она достигнет кошелька Боба в течение всего лишь нескольких секунд. Кошелек Боба немедленно идентифицирует транзакцию Алисы как входящий платеж, поскольку она содержит выход с возможностью погашения ключами Боба. Приложение кошелька Боба также может самостоятельно убедиться в правильности оформления транзакции. Если Боб пользуется своим собственным полным узлом, его кошелек может дополнительно удостовериться, что транзакция Алисы расходует только допустимые UTXO.

Майнинг биткойнов

Теперь транзакция Алисы распространяется в сети Биткойн. Она не станет частью *блокчейна* до тех пор, пока не будет включена в блок по итогам работы так называемого *майнинга* и пока этот блок не будет подтвержден всеми узлами. Подробное объяснение см. в главе 12.

Система защиты от подделок в Биткойне основана на вычислениях. Транзакции объединяются в блоки. Блоки имеют очень короткий заголовок, который необходимо создать определенным образом. Для этого требуется огромный объем вычислений, но для проверки их корректности достаточно совсем немного. Процесс майнинга в Биткойне выполняет две задачи:

- майнеры могут получать достойный доход только при создании блоков, которые соответствуют всем *правилам консенсуса* Биткойна. Поэтому обычно они заинтересованы во включении в свои блоки и блоки, на основе которых они создаются, только достоверных транзакций. Это дает пользователям возможность по своему усмотрению принимать на веру предположение о достоверности любой транзакции в блоке;
- майнинг в настоящее время создает новые биткойны в каждом блоке, подобно центральному банку, печатающему новые деньги. Количество биткойнов, создаваемых в каждом блоке, ограничено и со временем уменьшается, подчиняясь фиксированному графику эмиссии.

Майнинг обеспечивает оптимальный баланс между расходами и доходами. В процессе майнинга для решения вычислительной задачи расходуется электроэнергия. Успешный майнер получает за это вознаграждение в виде новых биткойнов и комиссии за транзакции. Однако вознаграждение будет получено только в том случае, если майнер включил лишь корректные транзакции, при этом корректность определяется правилами *консенсуса* протокола Биткойна. Этот тонкий баланс обеспечивает безопасность Биткойна без необходимости централизованного управления.

Технология майнинга построена по принципу децентрализованной лотереи. Каждый майнер может самостоятельно сформировать свой лотерейный билет путем создания *блока-кандидата* (*candidate block*). В этот блок входят новые транзакции, которые он хочет майнить, а также некоторые дополнительные поля данных. Майнер отправляет свой блок-кандидат в специально разработанный алгоритм, выполняющий скремблирование (или «хеширование») данных, и на выходе получаются данные, совершенно не похожие на исходные. Эта *хеш-функция* всегда будет выдавать один и тот же результат для одних и тех же входных данных, но никто не сможет предсказать результат для новых входных данных, даже если они совсем незначительно отличаются от предыдущих. Если выход хеш-функции совпадает с шаблоном по протоколу Биткойна, майнер выигрывает в лотерею, и пользователи Биткойна принимают блок с транзакциями в качестве корректного блока. Если результат не совпадает с шаблоном, майнер вносит небольшие изменения в свой блок-кандидат и повторяет попытку. На момент написания этой книги количество блоков-кандидатов, которые необходимо опробовать майнерам для получения выигрышной комбинации, составляло около 168 млрд трлн (миллиардов триллионов). Именно столько раз нужно запустить хеш-функцию.

Впрочем, после нахождения выигрышной комбинации любой желающий может убедиться в правильности блока, прогнав хеш-функцию всего один раз. Таким образом, для создания достоверного блока требуется невероятный объем работы, но проверка его достоверности требует лишь незначительных усилий. Простой процесс проверки позволяет с высокой степенью вероятности подтвердить факт проделанной работы, поэтому данные, необходимые для создания такого доказательства – в данном случае блок, называются *доказательством работы* (*Proof of Work*, PoW).

В новый блок добавляются транзакции, при этом приоритет отдается транзакциям с наиболее высокой комиссией и некоторым другим критериям. Все майнеры начинают процесс майнинга нового блока-кандидата транзакций сразу после поступления в сеть предыдущего блока, при этом они точно знают, что в этой итерации лотереи победил кто-то другой. Они сразу же создают новый блок-кандидат с привязкой к предыдущему блоку, заполняют его транзакциями и начинают вычислять PoW для него. Каждый майнер включает в блок-кандидат специальную транзакцию, которая выплачивает на его биткойн-адрес вознаграждение за блок плюс сумму транзакционных комиссий от всех включенных в блок-кандидат транзакций. Если они находят правильное решение, которое превращает блок-кандидат в достоверный блок, им выплачивается вознаграждение после внесения их успешного блока в глобальный блокчейн, и включенная ими транзакция становится расходной. Майнеры, участвующие в пуле майнинга, настраивают свои программы для создания блоков-кандидатов с назначением вознаграждения на адрес пула. После этого часть вознаграждения распределяется между членами майнерского пула пропорционально объему проделанной ими работы.

Транзакция Алисы была подхвачена сетью и включена в пул неподтвержденных транзакций. После проверки полноценным узлом она была включена в блок-кандидат. Примерно через пять минут после первой передачи транзакции из кошелька Алисы один из майнеров находит решение для блока и объяв-

ляет о нем в сети. После подтверждения победного блока другими майнерами они запускают новую лотерею для генерации следующего блока.

Победивший блок с транзакцией Алисы стал частью блокчейна. Блок, содержащий транзакцию Алисы, считается одним *подтверждением* этой транзакции. После распространения блока с транзакцией Алисы по сети создание альтернативного блока с другой версией транзакции Алисы (например, транзакции, которая не платит Бобу) потребует выполнения такого же объема работы, какой потребуется всем майнерам Биткойна для создания совершенно нового блока. Когда есть выбор из нескольких альтернативных блоков, полные узлы Биткойна выбирают цепочку достоверных блоков с наибольшим общим PoW, которую принято называть *лучшим блокчейном* (*best blockchain*). Чтобы вся сеть приняла альтернативный блок, необходимо майнить дополнительный новый блок поверх альтернативного.

Это означает, что у майнеров есть выбор. Они могут работать с Алисой над альтернативой транзакции, в которой она платит Бобу, например, чтобы Алиса платила майнерам часть денег, которые она ранее заплатила Бобу. Такое нечестное поведение потребует от них усилий по созданию двух новых блоков. Вместо этого майнеры, которые ведут себя честно, могут создать один новый блок и получить все вознаграждения от транзакций, которые они включают в него, плюс вознаграждение за блок. Обычно создание двух блоков за небольшую дополнительную плату гораздо менее выгодно, чем честное создание нового блока, поэтому вероятность намеренного изменения подтвержденной транзакции крайне невелика. Для Боба же это означает, что ему можно поверить в достоверность платежа от Алисы.

Здесь можно посмотреть блок с транзакцией Алисы: https://blockstream.info/block/0000000000000000027d39da52dd790d98f85895b02e764611cb7acf552e90.

Примерно через 19 минут после рассылки блока с транзакцией Алисы другой майнер добывает новый блок. Поскольку этот новый блок создан поверх блока с транзакцией Алисы (что дало транзакции Алисы уже два подтверждения), теперь транзакция Алисы может быть изменена только при условии добычи двух альтернативных блоков, а также нового блока, построенного поверх них. То есть чтобы Алиса смогла забрать отправленные Бобу деньги, потребуется добыть в общей сложности три блока. Каждый блок, добытый поверх блока с транзакцией Алисы, считается дополнительным подтверждением. По мере того как блоки накладываются друг на друга, отменить транзакцию становится все сложнее, а значит, у Боба появляется все больше уверенности в безопасности платежа Алисы.

На рис. 2.7 показан блок с транзакцией Алисы. Под ним находятся сотни тысяч блоков, связанных друг с другом в цепочку блоков (блокчейн) вплоть до блока № 0 (block #0), известного как *блок генезиса* (*genesis block*). Со временем, по мере увеличения «высоты» новых блоков, возрастает и сложность вычислений всей цепочки. Согласно определению, любой блок с более чем шестью подтверждениями считается особо трудноизменяемым, поскольку для пересчета шести блоков (плюс один новый блок) потребуется огромное количество вычислений. Более подробно процесс майнинга и способы укрепления доверия будут рассмотрены в главе 12.

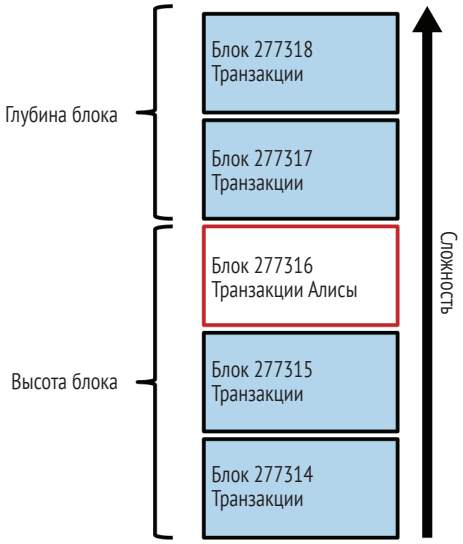


Рис. 2.7 ❖ Транзакция Алисы, включенная в блок

Расходование транзакции

Теперь, когда транзакция Алисы включена в блокчейн в качестве части блока, она видна всем приложениям сети Биткойн. Каждый полноценный узел Биткойна может независимо проверить транзакцию на достоверность и возможность расходования средств. Полноценные узлы проверяют каждый перевод средств с момента первого создания биткойнов в блоке и до каждой последующей транзакции, пока они не поступят на адрес Боба. Облегченные клиенты могут проводить частичную проверку платежей, убеждаясь в наличии транзакции в блокчейне и в существовании нескольких блоков, добытых после нее. Это дает гарантию того, что майнеры потратили значительные усилия на ее проведение (см. раздел «Облегченные клиенты»).

Теперь Боб имеет возможность тратить средства, полученные в результате этой и других транзакций. Например, он может заплатить подрядчику или поставщику, переведя средства, полученные от Алисы за подкаст, на счета этих новых владельцев. По мере расхода платежей, полученных от Алисы и других клиентов, Боб расширяет цепочку транзакций. Предположим, что Боб платит своему веб-дизайнеру Гопешу за новую страницу сайта. Теперь цепочка транзакций будет выглядеть как на рис. 2.8.

В этой главе было показано выстраивание транзакций в цепочку для перемещения средств от владельца к владельцу. Кроме того, была прослежена транзакция Алисы с момента ее создания в кошельке, прохождения через сеть Биткойн и до майнеров, которые записали ее в блокчейн. В остальных частях этой книги будут рассмотрены конкретные технологии, лежащие в основе кошельков, адресов, подписей, транзакций, сети и, наконец, майнинга.

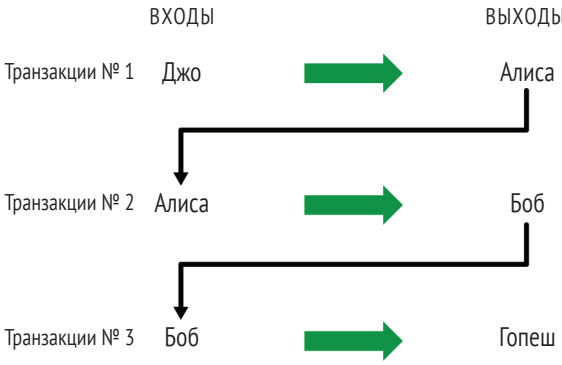


Рис. 2.8 ❖ Транзакция Алисы
как часть цепочки транзакций от Джо к Гопешу

Глава 3

Bitcoin Core: эталонная реализация

В обмен на свои ценные товары и услуги люди принимают деньги только в случае уверенности по поводу возможности их последующего расходования. Поддельные или внезапно обесценившиеся деньги уже невозможно потратить, поэтому каждый владелец биткойнов заинтересован в подтверждении подлинности полученных им средств. Система Биткойн спроектирована с учетом возможности использования программ, работающих исключительно на вашем локальном компьютере, для безупречного противодействия фальсификации, обесцениванию и ряду других критических проблем. Программы, выполняющие эту функцию, называют *узлами полной верификации* (*full verification node*), поскольку они проверяют каждую подтвержденную транзакцию в системе Биткойн на соответствие всем ее правилам. Узлы полной верификации, или просто *полноценные узлы* (*full nodes*), также предоставляют инструменты и данные для анализа работы системы Биткойн и всего, что происходит в его сети.

В этой главе описан процесс установки Bitcoin Core («Ядро Биткойн») – реализации, которую большинство операторов полноценных узлов используют с самого начала существования сети Биткойн. Затем будет проведен анализ блоков, транзакций и других данных вашего узла, которые являются достоверными – не потому, что какая-то влиятельная организация определила их таковыми, а потому, что ваш узел самостоятельно их верифицировал. На протяжении оставшейся части этой книги для создания и изучения данных, связанных с блокчейном и сетью, будет использоваться Bitcoin Core.

От Биткойна к Bitcoin Core

Концепция Биткойна является проектом *с открытым исходным кодом* (*open source*), и этот код доступен под открытой лицензией (MIT). Его можно свободно загружать и использовать в любых целях. Биткойн – это не просто проект с открытым исходным кодом, он также развивается силами открытого со-

общества добровольцев. Поначалу в него входил только сам Сатоши Накамото (Satoshi Nakamoto). К 2023 году над исходным кодом Биткойна работало более 1000 человек, при этом около десятка разработчиков трудились над кодом почти полный рабочий день, а еще несколько десятков – на условиях неполной занятости. Каждый может внести свой вклад в код, в том числе и вы!

Сатоши Накамото создал Биткойн, и программное обеспечение было в основном завершено до публикации технического описания (приведено в приложении А). Сатоши хотел убедиться в работоспособности реализации до публикации документа о ней. Та ранняя версия, известная в те времена как просто «Биткойн», впоследствии была сильно изменена и улучшена. В результате она превратилась в то, что сейчас принято называть *Bitcoin Core*, чтобы отличить ее от других реализаций. Концепция Bitcoin Core стала *эталонной реализацией* системы Биткойн. Это означает, что она служит образцом того, как должна быть реализована каждая часть технологии. На рис. 3.1 Эрика Ломброзо (Eric Lombrozo) показана архитектура Bitcoin Core.

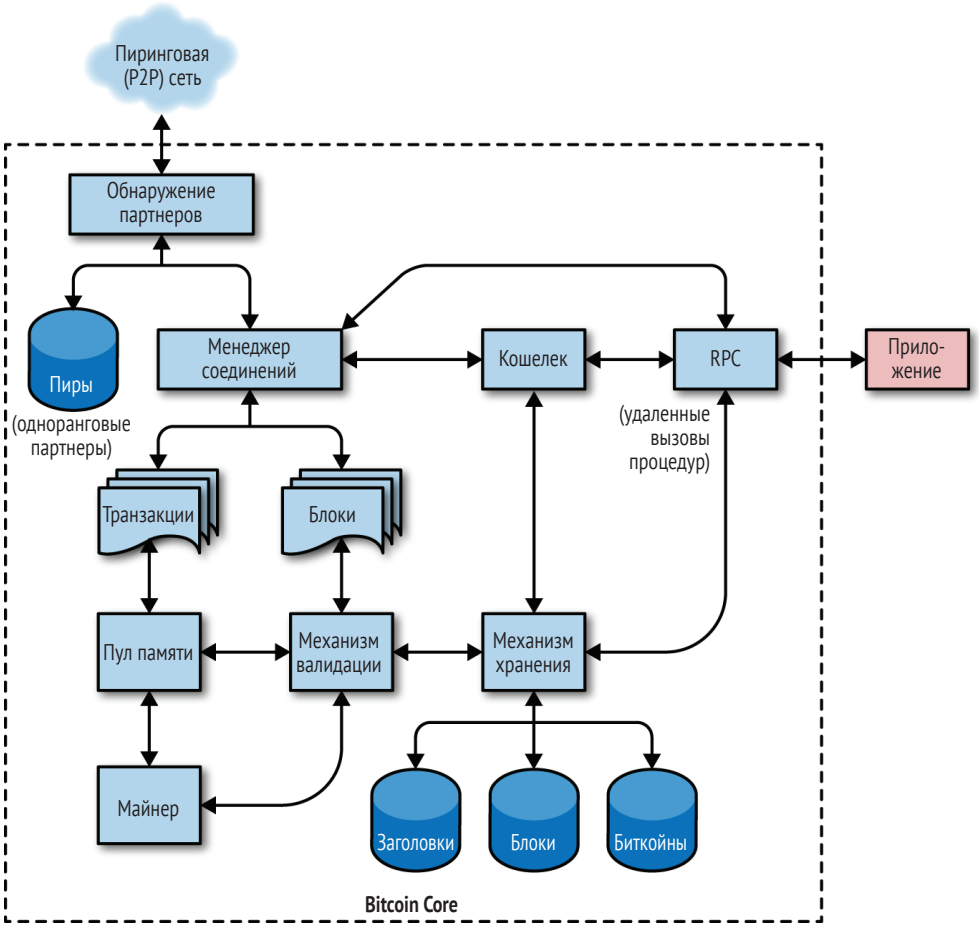


Рис. 3.1 ❖ Архитектура Bitcoin Core

Несмотря на использование Bitcoin Core в качестве эталонной реализации для многих основных частей системы, в техническом описании Bitcoin содержится информация о нескольких ранних компонентах системы. После 2011 года большинство основных элементов системы были задокументированы в наборах предложений по улучшению Биткойна (Bitcoin Improvement Proposals, BIPs, https://github.com/bitcoin/bips/tree/master). В этой книге спецификации BIP упоминаются под их номерами, например BIP9 описывает механизм для нескольких значительных обновлений системы Биткойн.

Среда разработки Биткойна

Разработчикам понадобится создать среду разработки со всеми инструментами, библиотеками и вспомогательным набором программ для написания приложений для Биткойна. В этой очень технической главе данный процесс будет рассмотрен шаг за шагом. Если материал окажется слишком сложным (и вы не собираетесь настраивать среду разработки), можно перейти к следующей, менее технической главе.

Компиляция Bitcoin Core из исходного кода

Исходный код Bitcoin Core можно загрузить в виде архива или путем клонирования исходного репозитория с GitHub. На странице загрузки Bitcoin Core (https://bitcoincore.org/bin/) необходимо выбрать самую последнюю версию и загрузить сжатый архив с исходным кодом. Также можно воспользоваться командной строкой Git для создания локальной копии исходного кода со страницы GitHub Bitcoin (https://github.com/bitcoin/bitcoin).

Во многих примерах этой главы используется интерфейс командной строки операционной системы, также известный как «оболочка» (shell), доступ к которому осуществляется через приложение «терминал». В командной строке отображается приглашение, пользователь вводит команду, в ответ появляется текст и новое приглашение для следующей команды. В вашей системе приглашение может выглядеть по-другому, поэтому в приведенных ниже примерах оно обозначается символом $. В примерах, когда после символа $ появляется текст, не набирайте символ $, а введите команду, следующую сразу за ним, затем нажмите **Enter**, чтобы выполнить ее. В этих примерах строки под каждой командой – это ответы операционной системы на эту команду. Когда появится следующий символ $, вы поймете, что это новая команда и вам следует повторить процесс.

Здесь используется команда `git` для создания локальной копии («клона») исходного кода:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Enumerating objects: 245912, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
```

```
remote: Total 245912 (delta 1), reused 2 (delta 1), pack-reused 245909
Receiving objects: 100% (245912/245912), 217.74 MiB | 13.05 MiB/s, done.
Resolving deltas: 100% (175649/175649), done.
```

Git – это наиболее популярная распределенная система учета версий, важная часть инструментария любого разработчика программного обеспечения. Вам может понадобиться установка поддержки команды `git` или ее графического интерфейса в вашей операционной системе, если у вас ее еще нет.

После завершения операции клонирования Git в каталоге *bitcoin* вашего компьютера появится полная локальная копия репозитория исходного кода. Перейдите в этот каталог с помощью команды `cd`:

```
$ cd bitcoin
```

Выбор версии Bitcoin Core

По умолчанию локальная копия будет синхронизирована с самой последней версией кода, которая может быть нестабильной или бета-версией Биткойна. Перед компиляцией кода необходимо выбрать определенную версию, проверив *тег* (*tag*) релиза. Это позволит синхронизировать локальную копию с конкретным снимком репозитория кода, обозначенным тегом с ключевым словом. Теги используются разработчиками для обозначения отдельных релизов кода по номеру версии. Для поиска доступных тегов используется команда `git tag`:

```
$ git tag
v0.1.5
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
v0.12.0rc1
v0.12.0rc2
...
```

В списке тегов отображаются все опубликованные версии Биткойна. Традиционно *релизы-кандидаты* (*release candidates*), предназначенные для тестирования, получают суффикс «rc». Стабильные релизы, которые можно запускать на рабочих системах, суффикса не имеют. Из приведенного выше списка следует выбрать релиз самой старшей версии, которой на момент написания книги была v24.0.1. Для синхронизации локального кода с этой версией можно использовать команду `git checkout`:

```
$ git checkout v24.0.1
Note: switching to 'v24.0.1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

HEAD is now at b3f866a8d Merge bitcoin/bitcoin#26647: 24.0.1 final changes
```

Вы можете подтвердить, что нужная версия «проверена» (checked out), введя команду `git status`:

```
HEAD detached at v24.0.1
nothing to commit, working tree clean
```

Настройка сборки Bitcoin Core

Исходный код содержит документацию в нескольких файлах. Основная документация находится в файле *README.md* каталога *bitcoin*. В этой главе будет выполнена сборка демона (daemon), то есть сервера (server) Bitcoin Core, который в системе Linux (Unix-подобная система) известен как `bitcoind`. Инструкции по компиляции клиента командной строки `bitcoind` на вашей платформе можно найти в файле *doc/build-unix.md*. Альтернативные инструкции можно найти в каталоге *doc*; например, *build-windows.md* содержит инструкции для Windows. На момент написания этой книги инструкции были доступны для Android, FreeBSD, NetBSD, OpenBSD, macOS (OSX), Unix и Windows.

Прежде чем приступить к компиляции Биткойна, следует внимательно изучить предварительные условия сборки, которые приведены в первой части инструкции по сборке. Речь о библиотеках, которые должны присутствовать в вашей системе до начала компиляции Биткойна. Если необходимые условия не будут соблюдены, процесс сборки завершится ошибкой. Если это случилось из-за невыполнения какого-либо предварительного условия, можно выполнить соответствующую установку, а затем возобновить процесс сборки с места остановки. Если все необходимые компоненты установлены, процесс сборки начнется с создания набора скриптов сборки с помощью скрипта `autogen.sh`:

```
$ ./autogen.sh
libtoolize: putting auxiliary files in AC_CONFIG_AUX_DIR, 'build-aux'.
libtoolize: copying file 'build-aux/ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'build-aux/m4'.
 ...
configure.ac:58: installing 'build-aux/missing'
src/Makefile.am: installing 'build-aux/depcomp'
parallel-tests: installing 'build-aux/test-driver'
```

Скрипт `autogen.sh` создает пакет автоматических конфигурационных скриптов, которые будут опрашивать вашу систему для определения правильных настроек и проверки наличия всех необходимых библиотек для компиляции кода. Наиболее важным из них является скрипт `configure`, который предоставляет множество различных опций для настройки процесса сборки. Для просмотра различных опций можно использовать флаг `--help`:

```
$ ./configure --help
`configure' configures Bitcoin Core 24.0.1 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

...
Optional Features:
```

```
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
  --enable-silent-rules   less verbose build output (undo: "make V=1")
  --disable-silent-rules  verbose build output (undo: "make V=0")
...
```

Скрипт `configure` позволяет включить или отключить определенные функции `bitcoind` с помощью флагов `--enable-FEATURE` и `--disable-FEATURE`, где `FEATURE` заменяется именем функции, как указано в справке. В этой главе будет создан клиент `bitcoind` со всеми функциями по умолчанию. Флаги конфигурации использоваться не будут, но с ними следует ознакомиться, чтобы понимать, какие дополнительные функции входят в состав клиента. В случае если вы работаете в учебном заведении, ограничения компьютерного класса могут заставить вас устанавливать приложения в свой собственный домашний каталог (например, используя `--prefix=$HOME`).

Вот несколько полезных опций, которые переопределяют действия скрипта `configure` по умолчанию:

`--prefix=$HOME`
: Эта команда переопределяет место установки по умолчанию (а именно */usr/local/*) для итогового исполняемого файла. Используйте `$HOME` для размещения всех файлов в вашем домашнем каталоге или в другом месте.

`--disable-wallet`
: Используется для отключения эталонной реализации кошелька.

`--with-incompatible-bdb`
: При сборке кошелька разрешает использовать несовместимую версию библиотеки Berkeley DB.

`--with-gui=no`
: Не создавать графический интерфейс пользователя, для которого требуется библиотека Qt. Это позволяет собрать только сервер и командную строку Bitcoin Core.

Далее следует запустить скрипт configure, который автоматически обнаружит все необходимые библиотеки и создаст настраиваемый скрипт сборки для вашей системы:

```
$ ./configure
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
...
[далее следует несколько страниц с результатами тестов конфигурации]
...
```

Если все пройдет успешно, команда `configure` завершит работу созданием пользовательских скриптов сборки, которые позволят скомпилировать `bitcoind`. Если не хватает библиотек или есть ошибки, команда configure завер-

шится ошибкой, а не созданием скриптов сборки. Если возникает ошибка, то, скорее всего, это связано с отсутствием или несовместимостью библиотек. Нужно еще раз просмотреть документацию по сборке и проверить, что все необходимые компоненты установлены. Затем нужно снова запустить `configure` и посмотреть, устранилась ли ошибка.

Сборка исполняемых файлов Bitcoin Core

Далее необходимо выполнить компиляцию исходного кода – этот процесс может занять до часа, в зависимости от быстродействия вашего процессора и объема доступной памяти. Если возникнет ошибка или процесс компиляции будет прерван, его можно возобновить в любое время, введя `make` еще раз. Чтобы начать компиляцию исполняемого приложения, нужно набрать **`make`**:

```
$ make
Making all in src
  CXX      bitcoind-bitcoind.o
  CXX      libbitcoin_node_a-addrdb.o
  CXX      libbitcoin_node_a-addrman.o
  CXX      libbitcoin_node_a-banman.o
  CXX      libbitcoin_node_a-blockencodings.o
  CXX      libbitcoin_node_a-blockfilter.o
[... множество сообщений о ходе компиляции ...]
```

На быстрой системе с более чем одним процессором можно задать количество параллельных запусков компиляции. Например, `make -j 2` будет использовать два ядра в случае их наличия. Если все прошло успешно, компиляция Bitcoin Core завершена. Необходимо запустить набор модульных тестов с помощью `make check`, чтобы убедиться, что скомпилированные библиотеки не имеют очевидных ошибок. Последний шаг – установка различных исполняемых файлов в вашу систему с помощью команды `make install`. Может потребоваться ввод пароля пользователя, поскольку этот шаг требует прав администратора:

```
$ make check && sudo make install
Password:
Making install in src
 ../build-aux/install-sh -c -d '/usr/local/lib'
libtool: install: /usr/bin/install -c bitcoind /usr/local/bin/bitcoind
libtool: install: /usr/bin/install -c bitcoin-cli /usr/local/bin/bitcoin-cli
libtool: install: /usr/bin/install -c bitcoin-tx /usr/local/bin/bitcoin-tx
...
```

При установке по умолчанию `bitcoind` размещается в */usr/local/bin*. Убедиться в правильности установки Bitcoin Core можно с помощью запроса к системе о пути к исполняемым файлам, как показано ниже:

```
$ which bitcoind
/usr/local/bin/bitcoind

$ which bitcoin-cli
/usr/local/bin/bitcoin-cli
```

Запуск узла Bitcoin Core

Пиринговая сеть Биткойн состоит из сетевых «узлов» (nodes), которыми управляют преимущественно частные лица, а также некоторые компании, оказывающие услуги по работе с биткойнами. Владельцы узлов сети Биткойн имеют прямой и полномочный доступ к блокчейну Биткойна, располагая при этом локальной копией всех расходуемых биткойнов, прошедшей независимое подтверждение в их собственной системе. Если вы управляете своим узлом, то для подтверждения транзакции вам больше не понадобится привлекать третьих лиц. Кроме того, использование узла Биткойна для полной проверки транзакций, которые вы получаете на свой кошелек, вносит свой вклад в развитие сети Биткойн и помогает сделать ее более надежной.

Однако запуск узла требует первоначальной загрузки и обработки более 500 ГБайт данных и около 400 МБайт транзакций Биткойна ежедневно. Эти показатели указаны по состоянию на 2023 год и, скорее всего, со временем будут расти. В случае прекращения работы узла или отключения от интернета на несколько дней ему потребуется загрузить все пропущенные данные. Например, если закрыть Bitcoin Core на 10 дней, то при следующем запуске потребуется загрузить порядка 4 ГБайт.

Если вы намерены индексировать все транзакции и хранить полную копию блокчейна, то в зависимости от предпочтений может понадобиться много дискового пространства – как минимум 1 ТБайт при планируемой работе Bitcoin Core в течение нескольких лет. По умолчанию узлы Биткойна также передают транзакции и блоки другим узлам (так называемым «пирам» – peers), что требует высокой пропускной способности интернета. Если ваш интернет-канал лимитирован по скорости, объему данных или тарифицируется по гигабитам, возможно, не стоит подключать к нему узел Биткойна или запускать его с ограничением пропускной способности (см. раздел «Настройка узла Bitcoin Core»). Вместо этого можно подключить узел к альтернативной сети, например к бесплатному спутниковому провайдеру данных Blockstream Satellite (https://blockstream.com/satellite/).

По умолчанию Bitcoin Core хранит полную копию блокчейна с почти каждой транзакцией, когда-либо подтвержденной в сети Биткойн с момента ее создания в 2009 году. Объем этого массива данных составляет сотни гигабайт, и он загружается постепенно в течение нескольких часов или дней, в зависимости от скорости вашего процессора и интернет-соединения. Система Bitcoin Core не сможет обрабатывать транзакции или обновлять балансы счетов до тех пор, пока не будет загружен полный массив данных блокчейна. Поэтому необходимо предусмотреть достаточное количество дискового пространства, пропускной способности и времени для завершения первоначальной синхронизации. Можно настроить Bitcoin Core на уменьшение размера блокчейна путем удаления старых блоков, но он все равно должен загрузить полный массив данных.

Несмотря на все эти требования к используемым ресурсам, узлы Биткойна создают тысячи людей. Некоторые из них работают на таких простых системах, как Raspberry Pi (компьютер стоимостью $35 и размером с колоду игральных карт).

Для чего вам может понадобиться запуск узла? Вот несколько наиболее распространенных причин:

- вы не хотите полагаться на сторонних лиц для подтверждения получаемых вами транзакций;
- вы не хотите раскрывать посторонним лицам информацию о принадлежащих вашему кошельку транзакциях;
- вы разрабатываете программное обеспечение для Биткойна и нуждаетесь в узле Биткойна для программируемого (API) доступа к сети и блокчейну;
- вы создаете приложения, которые должны подтверждать транзакции в соответствии с правилами консенсуса Биткойна. Обычно компании, создающие программы для Биткойна, управляют несколькими узлами;
- вы хотите поддержать Биткойн. Запуск узла, который вы используете для подтверждения поступающих на ваш кошелек транзакций, делает сеть более стабильной.

Если вы читаете эту книгу и заинтересованы в обеспечении надежной защиты, конфиденциальности или разработке программного обеспечения для Биткойна, вам стоит запустить свой собственный узел.

Настройка узла Bitcoin Core

При каждом запуске Bitcoin Core будет выполнять поиск файла конфигурации в каталоге данных. В этом разделе будут рассмотрены различные параметры конфигурации и создан файл конфигурации. Для поиска файла конфигурации необходимо запустить в терминале команду `bitcoind -printtoconsole` и найти первые пару строк:

```
$ bitcoind -printtoconsole
2023-01-28T03:21:42Z Bitcoin Core version v24.0.1
2023-01-28T03:21:42Z Using the 'x86_shani(1way,2way)' SHA256 implementation
2023-01-28T03:21:42Z Using RdSeed as an additional entropy source
2023-01-28T03:21:42Z Using RdRand as an additional entropy source
2023-01-28T03:21:42Z Default data directory /home/harding/.bitcoin
2023-01-28T03:21:42Z Using data directory /home/harding/.bitcoin
2023-01-28T03:21:42Z Config file: /home/harding/.bitcoin/bitcoin.conf
...
[много дополнительной информации по отладке]
...
```

После определения местоположения файла конфигурации можно нажать **Ctrl+C** для выключения узла. Обычно файл конфигурации находится в каталоге данных *.bitcoin* домашнего пользовательского каталога. Откройте файл конфигурации в удобном для вас редакторе.

Bitcoin Core предлагает более 100 опций конфигурирования, которые изменяют характер работы узла сети, хранение блокчейна и многие другие аспекты его работы. Для просмотра списка этих опций можно выполнить команду `bitcoind --help`:

```
$ bitcoind --help
Bitcoin Core version v24.0.1

Usage:  bitcoind [options]                     Start Bitcoin Core

Options:

  -?
       Print this help message and exit

-alertnotify=<cmd>
       Execute command when an alert is raised (%s in cmd is replaced by message)
...
[множество других опций]
```

Ниже приведены наиболее важные опции, которые можно задать в конфигурационном файле или в качестве параметров командной строки для `bitcoind`:

`alertnotify`
: Запуск указанной команды или скрипта для отправки экстренных оповещений владельцу этого узла.

`conf`
: Альтернативное расположение конфигурационного файла. Имеет смысл только в качестве параметра командной строки для `bitcoind`, так как не может находиться внутри конфигурационного файла, на который ссылается.

`datadir`
: Выбор каталога и файловой системы, где будут храниться все данные блокчейна. По умолчанию это подкаталог `.bitcoin` в вашем основном каталоге. В зависимости от вашей конфигурации он может занимать от 10 Гбайт до почти 1 Тбайт (на момент написания книги), а максимальный размер, как ожидается, будет увеличиваться на несколько сотен гигабайт в год.

`prune`
: Сокращение дискового пространства блокчейна до указанного количества мегабайтов за счет удаления старых блоков. Используется на узлах с ограниченными ресурсами, где не помещается полный блокчейн. Остальные компоненты системы будут использовать другое дисковое пространство, которое не может быть сокращено в данный момент, поэтому все равно потребуется хотя бы минимальное количество места, указанное в опции `datadir`.

`txindex`
: Ведение индекса всех транзакций. Это позволит получить любую транзакцию программным путем по ее идентификатору, при условии что блок с этой транзакцией не был сокращен.

`dbcache`
: Размер кеша UTXO. По умолчанию он составляет 450 мегабайт (Мбайт). Можно увеличить этот объем на высокопроизводительном оборудовании для менее частого выполнения чтения и записи с диска или уменьшить на малопроизводительном оборудовании для экономии памяти за счет более частого использования диска.

`blocksonly`
: Сокращение нагрузки на канал связи за счет приема от пиров только блоков подтвержденных транзакций, без передачи неподтвержденных транзакций.

`maxmempool`
: Ограничение пула памяти транзакций указанным количеством мегабайтов. Используется для снижения загрузки памяти на узлах с ограниченным объемом памяти.

Индекс базы данных транзакций и опция txindex

По умолчанию Bitcoin Core создает базу данных, которая хранит только транзакции по кошельку пользователя. Если вам нужно иметь доступ к любой транзакции с помощью команд вроде getrawtransaction (см. раздел «Исследование и декодирование транзакций»), необходимо настроить Bitcoin Core на формирование полного индекса транзакций. Это может быть сделано с помощью опции txindex. В конфигурационном файле Bitcoin Core нужно установить txindex=1. Если изначально этот параметр не задан, а позже будет установлено полное индексирование, придется подождать завершения перенастройки индекса.

В примере 3.1 приведено сочетание предыдущих вариантов с полностью индексированным узлом, который используется в качестве API-бэкенда для биткойн-приложения.

Пример 3.1 ❖ Пример конфигурации полностью проиндексированного узла

```
alertnotify=myemailscript.sh "Alert: %s"
datadir=/lotsofspace/bitcoin
txindex=1
```

В примере 3.2 показан узел с ограниченными ресурсами, который работает на сервере меньшей мощности.

Пример 3.2 ❖ Пример конфигурации системы с ограниченными ресурсами

```
alertnotify=myemailscript.sh "Alert: %s"
blocksonly=1
prune=5000
dbcache=150
maxmempool=150
```

После редактирования конфигурационного файла и установки необходимых опций можно протестировать `bitcoind` с этой конфигурацией. Для работы в активном режиме с выводом данных на экран консоли запустите Bitcoin Core с опцией `printtoconsole`:

```
$ bitcoind -printtoconsole
2023-01-28T03:43:39Z Bitcoin Core version v24.0.1
2023-01-28T03:43:39Z Using the 'x86_shani(1way,2way)' SHA256 implementation
2023-01-28T03:43:39Z Using RdSeed as an additional entropy source
2023-01-28T03:43:39Z Using RdRand as an additional entropy source
2023-01-28T03:43:39Z Default data directory /home/harding/.bitcoin
2023-01-28T03:43:39Z Using data directory /lotsofspace/bitcoin
2023-01-28T03:43:39Z Config file: /home/harding/.bitcoin/bitcoin.conf
```

```
2023-01-28T03:43:39Z Config file arg: [main] blockfilterindex="1"
2023-01-28T03:43:39Z Config file arg: [main] maxuploadtarget="1000"
2023-01-28T03:43:39Z Config file arg: [main] txindex="1"
2023-01-28T03:43:39Z Setting file arg: wallet = ["msig0"]
2023-01-28T03:43:39Z Command-line arg: printtoconsole=""
2023-01-28T03:43:39Z Using at most 125 automatic connections
2023-01-28T03:43:39Z Using 16 MiB out of 16 MiB requested for signature cache
2023-01-28T03:43:39Z Using 16 MiB out of 16 MiB requested for script execution
2023-01-28T03:43:39Z Script verification uses 3 additional threads
2023-01-28T03:43:39Z scheduler thread start
2023-01-28T03:43:39Z [http] creating work queue of depth 16
2023-01-28T03:43:39Z Using random cookie authentication.
2023-01-28T03:43:39Z Generated RPC cookie /lotsofspace/bitcoin/.cookie
2023-01-28T03:43:39Z [http] starting 4 worker threads
2023-01-28T03:43:39Z Using wallet directory /lotsofspace/bitcoin/wallets
2023-01-28T03:43:39Z init message: Verifying wallet(s)…
2023-01-28T03:43:39Z Using BerkeleyDB version Berkeley DB 4.8.30
2023-01-28T03:43:39Z Using /16 prefix for IP bucketing
2023-01-28T03:43:39Z init message: Loading P2P addresses…
2023-01-28T03:43:39Z Loaded 63866 addresses from peers.dat 114ms
[... больше стартовых сообщений ...]
```

Убедившись в корректности загрузки настроек и правильности работы процесса, можно прервать его нажатием **Ctrl-C**.

Для работы Bitcoin Core в фоновом режиме в качестве процесса запустите его с параметром `daemon`, как `bitcoind -daemon`.

Чтобы отслеживать процесс и статус вашего узла Биткойна, запустите его в режиме демона (daemon mode), а затем выполните команду `bitcoin-cli getblockchaininfo`:

```
$ bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 0,
  "headers": 83999,
  "bestblockhash": "[...]19d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "difficulty": 1,
  "time": 1673379796,
  "mediantime": 1231006505,
  "verificationprogress": 3.783041623201835e-09,
  "initialblockdownload": true,
  "chainwork": "[...]0000000000000000000000000000000000000000000000000100010001",
  "size_on_disk": 89087,
  "pruned": false,
  "warnings": ""
}
```

Здесь показан узел с высотой блокчейна в 0 блоков и 83 999 заголовками. Сначала узел получает заголовки блоков от своих пиров для определения блокчейна с наибольшим количеством доказательств работы, а затем продолжает загружать полные блоки, проверяя их по мере поступления.

Когда выбранные параметры конфигурации окажутся приемлемыми, следует добавить Bitcoin Core в стартовые скрипты операционной системы, чтобы

он загружался регулярно и возобновлял работу при перезапуске операционной системы. В каталоге исходных текстов Bitcoin Core в разделе *contrib/init* можно найти несколько примеров стартовых скриптов для различных операционных систем, а также файл *README.md* с информацией об использовании того или иного скрипта в конкретной системе.

API Bitcoin Core

В Bitcoin Core используется интерфейс JSON-RPC, доступ к которому также можно получить с помощью помощника командной строки `bitcoin-cli`. Командная строка позволяет экспериментировать со всеми функциями в интерактивном режиме. Эти функции также доступны через программный API. Для начала, чтобы увидеть список доступных RPC-команд Bitcoin Core, нужно ввести команду `help`:

```
$ bitcoin-cli help
+== Blockchain ==
getbestblockhash
getblock "blockhash" ( verbosity )
getblockchaininfo
...
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
walletprocesspsbt "psbt" ( sign "sighashtype" bip32derivs finalize )
```

Каждая из этих команд может иметь несколько параметров. Для получения дополнительной справки, подробного описания и информации о параметрах следует добавить имя команды после help. Например, чтобы просмотреть справку по RPC-команде `getblockhash`:

```
$ bitcoin-cli help getblockhash
getblockhash height

Returns hash of block in best-block-chain at height provided.
(Возвращает хеш-значение блока в самой правильной цепочке по заданному индексу).

Arguments:
(Аргументы:)
1. height     (numeric, required) The height index
(1. Высота (число, обязательный)(Высота блока)

Result:
(Результат:)
"hex"     (string) The block hash
("16-ричный" (строка) (хеш блока)

Examples:
(Примеры:)
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id": "curltest",
  "method": "getblockhash",
  "params": [1000]}' -H 'content-type: text/plain;' http://127.0.0.1:8332/
```

В конце справочной информации приведены два примера выполнения команды RPC с помощью вспомогательного модуля `bitcoin-cli` или HTTP-клиента `curl`. Эти примеры показывают возможные способы запуска команды. Скопируйте первый пример и посмотрите на результат:

```
$ bitcoin-cli getblockhash 1000
00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09
```

В результате получается хеш блока, о котором более подробно будет рассказано в следующих главах. А пока эта команда должна выдать тот же результат в вашей системе, подтверждая факт работы узла Bitcoin Core, приема команд и наличия информации о блоке 1000 для выдачи результатов вам.

В следующих разделах будут показаны некоторые очень полезные RPC-команды и предполагаемые результаты их выполнения.

Сбор информации о состоянии Bitcoin Core

В Bitcoin Core отчеты о состоянии различных модулей можно получить через интерфейс JSON-RPC. К наиболее важным командам относятся `getblockchaininfo`, `getmempool info`, `getnetworkinfo` и `getwalletinfo`.

RPC-команда `getblockchaininfo` для Биткойна уже была представлена ранее. Команда `getnetwork info` отображает основную информацию о состоянии узла сети Биткойн. Для ее выполнения можно использовать `bitcoin-cli`:

```
$ bitcoin-cli getnetworkinfo
```

```
{
  "version": 240001,
  "subversion": "/Satoshi:24.0.1/",
  "protocolversion": 70016,
  "localservices": "0000000000000409",
  "localservicesnames": [
    "NETWORK",
    "WITNESS",
    "NETWORK_LIMITED"
  ],
  "localrelay": true,
  "timeoffset": -1,
  "networkactive": true,
  "connections": 10,
  "connections_in": 0,
  "connections_out": 10,
  "networks": [
    "...detailed information about all networks..."
  ],
  "relayfee": 0.00001000,
  "incrementalfee": 0.00001000,
  "localaddresses": [
  ],
  "warnings": ""
}
```

Данные возвращаются в формате JavaScript Object Notation (JSON), который легко «проглатывают» все языки программирования, и при этом он вполне читаем человеком. В числе этих данных можно видеть номера версий программного обеспечения Bitcoin Core и протокола Биткойн. Кроме того, можно увидеть текущее количество соединений и различную информацию о сети Биткойн и настройках этого узла.

Чтобы нагнать нынешнюю высоту блокчейна, у `bitcoind` уйдет некоторое время, возможно не один день, поскольку он загружает блоки с других узлов Биткойна и проверяет каждую транзакцию в этих блоках – на момент написания книги их было почти миллиард. Чтобы проверить ход работы и количество известных блоков, можно воспользоваться функцией `getblockchaininfo`. В примерах этой главы предполагается, что вы находитесь как минимум на блоке 775 072. Поскольку безопасность транзакций Биткойна зависит от блоков, некоторая информация в следующих примерах будет немного меняться в зависимости от количества блоков у вашего узла.

Исследование и декодирование транзакций

В разделе «Покупка в интернет-магазине» Алиса сделала покупку в магазине Боба. Ее транзакция была записана в блокчейн. С помощью API можно отыскать и изучить эту транзакцию, передав в качестве параметра идентификатор этой транзакции (txid):

```
$ bitcoin-cli getrawtransaction 466200308696215bbc949d5141a49a41\
38ecdfdfaa2a8029c1f9bcecd1f96177

01000000000101eb3ae38f27191aa5f3850dc9cad00492b88b72404f9da13569
8679268041c54a0100000000ffffffff02204e0000000000002251203b41daba
4c9ace578369740f15e5ec880c28279ee7f51b07dca69c7061e07068f8240100
0000000001600147752c165ea7be772b2c0acb7f4d6047ae6f4768e0141cf5efe
2d8ef13ed0af21d4f4cb82422d6252d70324f6f4576b727b7d918e521c00b51b
e739df2f899c49dc267c0ad280aca6dab0d2fa2b42a45182fc83e81713010000
0000
```

Идентификатор транзакции (txid) не является авторизующей информацией. Отсутствие txid в блокчейне не означает, что транзакция не была обработана. Это называется «пластичностью транзакции» (transaction malleability), поскольку до подтверждения в блоке она может быть изменена, что влечет за собой изменение ее txid. После включения транзакции в блокчейн ее txid не может измениться, если только не произойдет реорганизация блокчейна, в результате которой этот блок будет удален из лучшего блокчейна. После нескольких подтверждений транзакции реорганизации происходят крайне редко.

Команда `getrawtransaction` выдает последовательную транзакцию в шестнадцатеричном виде. Для декодирования этих данных можно воспользоваться командой `decoderawtransaction`, передав шестнадцатеричные данные в качестве параметра. Можно скопировать шестнадцатеричный код, выданный командой `getrawtransaction`, и вставить его в качестве параметра в команду `decoderawtransaction`:

```
$ bitcoin-cli decoderawtransaction 01000000000101eb3ae38f27191aa5f3850dc9cad0\
0492b88b72404f9da135698679268041c54a0100000000ffffffff02204e00000000000022512\
03b41daba4c9ace578369740f15e5ec880c28279ee7f51b07dca69c7061e07068f82401000000\
```

```
00001600147752c165ea7be772b2c0acb7f4d6047ae6f4768e0141cf5efe2d8ef13ed0af21d4f\
4cb82422d6252d70324f6f4576b727b7d918e521c00b51be739df2f899c49dc267c0ad280aca6\
dab0d2fa2b42a45182fc83e817130100000000
{
  "txid": "466200308696215bbc949d5141a49a4138ecdfdfaa2a8029c1f9bcecd1f96177",
  "hash": "f7cdbc7cf8b910d35cc69962e791138624e4eae7901010a6da4c02e7d238cdac",
  "version": 1,
  "size": 194,
  "vsize": 143,
  "weight": 569,
  "locktime": 0,
  "vin": [
    {
      "txid": "4ac541802679866935a19d4f40728bb89204d0cac90d85f3a51a19...aeb",
      "vout": 1,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "txinwitness": [
        "cf5efe2d8ef13ed0af21d4f4cb82422d6252d70324f6f4576b727b7d918e5...301"
      ],
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.00020000,
      "n": 0,
      "scriptPubKey": {
        "asm": "1 3b41daba4c9ace578369740f15e5ec880c28279ee7f51b07dca...068",
        "desc": "rawtr(3b41daba4c9ace578369740f15e5ec880c28279ee7f51b...6ev",
        "hex": "51203b41daba4c9ace578369740f15e5ec880c28279ee7f51b07d...068",
        "address": "bc1p8dqa4wjvnt890qmfws83te0v3qxzsfu7ul63kp7u56w8q...5qn",
        "type": "witness_v1_taproot"
      }
    },
    {
     "value": 0.00075000,
     "n": 1,
     "scriptPubKey": {
       "asm": "0 7752c165ea7be772b2c0acb7f4d6047ae6f4768e",
       "desc": "addr(bc1qwafvze0200nh9vkq4jmlf4sy0tn0ga5w0zpkpg)#qq404gts",
       "hex": "00147752c165ea7be772b2c0acb7f4d6047ae6f4768e",
       "address": "bc1qwafvze0200nh9vkq4jmlf4sy0tn0ga5w0zpkpg",
       "type": "witness_v0_keyhash"
      }
    }
  ]
}
```

Декодирование транзакции показывает все ее компоненты, включая входы и выходы. В данном случае можно увидеть, что транзакция использовала один вход и сгенерировала два выхода. Входом для этой транзакции был выход из

ранее подтвержденной транзакции (показан как вход `txid`). Два выхода соответствуют платежу Бобу и возврату сдачи Алисе.

Можно продолжить исследование блокчейна, изучив предыдущую транзакцию, на которую ссылается ее `txid` в этой транзакции, с помощью тех же команд (например, `getrawtransaction`). Переходя от транзакции к транзакции, можно проследить цепочку транзакций, по которой происходит передача средств от одного владельца к другому.

Изучение блоков

Изучение блоков схоже с анализом транзакций. Однако на блоки можно ссылаться как по *высоте* блока, так и по его *хешу*. Для начала нужно найти блок по его высоте. Для этого используется команда `getblockhash`, которая принимает в качестве параметра высоту блока и возвращает *хеш заголовка* (header hash) этого блока:

```
$ bitcoin-cli getblockhash 123456
000000000000002917ed80650c6174aac8dfc46f5fe36480aaef682ff6cd83c3ca
```

Теперь, когда известен хеш заголовка выбранного блока, можно сделать запрос по нему. Для этого используется команда `getblock` с хешем блока в качестве параметра:

```
$ bitcoin-cli getblock 000000000000002917ed80650c6174aac8dfc46f5fe36480aaef682f\
f6cd83c3ca

{
  "hash": "000000000000002917ed80650c6174aac8dfc46f5fe36480aaef682ff6cd83c3ca",
  "confirmations": 651742,
  "height": 123456,
  "version": 1,
  "versionHex": "00000001",
  "merkleroot": "0e60651a9934e8f0decd1c[...]48fca0cd1c84a21ddfde95033762d86c",
  "time": 1305200806,
  "mediantime": 1305197900,
  "nonce": 2436437219,
  "bits": "1a6a93b3",
  "difficulty": 157416.4018436489,
  "chainwork": "[...]0000000000000000000000000000000000000000541788211ac227bc",
  "nTx": 13,
  "previousblockhash": "[...]60bc96a44724fd72daf9b92cf8ad00510b5224c6253ac40095",
  "nextblockhash": "[...]00129f5f02be247070bf7334d3753e4ddee502780c2acaecec6d66",
  "strippedsize": 4179,
  "size": 4179,
  "weight": 16716,
  "tx": [
    "5b75086dafeede555fc8f9a810d8b10df57c46f9f176ccc3dd8d2fa20edd685b",
    "e3d0425ab346dd5b76f44c222a4bb5d16640a4247050ef82462ab17e229c83b4",
    "137d247eca8b99dee58e1e9232014183a5c5a9e338001a0109df32794cdcc92e",
    "5fd167f7b8c417e59106ef5acfe181b09d71b8353a61a55a2f01aa266af5412d",
    "60925f1948b71f429d514ead7ae7391e0edf965bf5a60331398dae24c6964774",
    "d4d5fc1529487527e9873256934dfb1e4cdcb39f4c0509577ca19bfad6c5d28f",
```

```
    "7b29d65e5018c56a33652085dbb13f2df39a1a9942bfe1f7e78e97919a6bdea2",
    "0b89e120efd0a4674c127a76ff5f7590ca304e6a064fbc51adffbd7ce3a3deef",
    "603f2044da9656084174cfb5812feaf510f862d3addcf70cacce3dc55dab446e",
    "9a4ed892b43a4df916a7a1213b78e83cd83f5695f635d535c94b2b65ffb144d3",
    "dda726e3dad9504dce5098dfab5064ecd4a7650bfe854bb2606da3152b60e427",
    "e46ea8b4d68719b65ead930f07f1f3804cb3701014f8e6d76c4bdbc390893b94",
    "864a102aeedf53dd9b2baab4eeb898c5083fde6141113e0606b664c41fe15e1f"
  ]
}
```

Запись подтверждения сообщает глубину этого блока – сколько блоков было построено поверх него. Это указывает на сложность изменения любой транзакции в этом блоке. Высота говорит нам о количестве блоков, предшествовавших этому. Здесь же отображается версия блока, время его создания (по данным майнера), среднее время 11 блоков, предшествующих этому блоку (майнерам сложно изменить показатели времени), а также размер блока в трех различных показателях (унаследованный урезанный размер, полный размер и размер в условных единицах). Здесь также представлены поля, используемые для обеспечения безопасности и доказательства работы (корень Меркла, нонс, биты, сложность и цепочки); подробнее о них будет рассказано в главе 12.

Использование программного интерфейса Bitcoin Core

Вспомогательная программа `bitcoin-cli` очень полезна для знакомства с API Bitcoin Core и тестирования его функций. Но суть API заключается в программном доступе к функциям. В этом разделе на примере Bitcoin Core будет показан механизм доступа к нему из другой программы.

В API Bitcoin Core используется интерфейс JSON-RPC. Формат JSON очень удобен для представления данных, которые без труда читают как люди, так и программы. Аббревиатура RPC означает удаленный вызов процедур (Remote Procedure Call). То есть через сетевой протокол происходит вызов удаленных (на узле Bitcoin Core) процедур (функций). В данном случае используется протокол HTTP.

При применении `bitcoin-cli` для получения справки о команде программа показала пример использования `curl`, универсального HTTP-клиента командной строки, для создания одного из таких вызовов JSON-RPC:

```
$ curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest",
  "method": "getblockchaininfo",
  "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:8332/
```

Здесь видно, что `curl` отправляет HTTP-запрос на локальный хост (127.0.0.1), подключается к стандартному порту Bitcoin RPC (8332) и отправляет запрос `jsonrpc` для использования метода `getblockchaininfo` в кодировке text/plain.

Можно заметить, что `curl` будет запрашивать учетные данные для отправки вместе с этим запросом. При каждом запуске Bitcoin Core будет создавать случайный пароль и размещать его в каталоге данных под названием `.cookie`. Вспомогательная программа `bitcoin-cli` может прочитать этот файл с паролем из каталога данных. Точно так же можно скопировать пароль и передать его

curl (или любым другим программам Bitcoin Core RPC более высокого уровня), как показано в примере 3.3.

Пример 3.3 ❖ Аутентификация с помощью файлов cookie в Bitcoin Core

```
$ cat .bitcoin/.cookie
  __cookie__:17c9b71cef21b893e1a019f4bc071950c7942f49796ed061b274031b17b19cd0

$ curl
  --user __cookie__:17c9b71cef21b893e1a019f4bc071950c7942f49796ed061b274031b17b19cd0
  --data-binary '{"jsonrpc": "1.0", "id":"curltest",
  "method": "getblockchaininfo",
  "params": [] }' -H 'content-type: text/plain;' http://127.0.0.1:8332/

{"result":{"chain":"main","blocks":799278,"headers":799278,
"bestblockhash":"0000000000000000000018387c50988ec705a95d6f765b206b6629971e6978879",
"difficulty":53911173001054.59,"time":1689703111,"mediantime":1689701260,
"verificationprogress":0.9999979206082515,"initialblockdownload":false,
"chainwork":"0000000000000000000000000000000000000000004f3e111bf32bcb47f9dfad5b",
"size_on_disk":563894577967,"pruned":false,"warnings":""},"error":null,
"id":"curltest"}
```

В качестве альтернативы можно задать статичный пароль с помощью вспомогательного скрипта, представленного в папке *./share/rpcauth/rpcauth.py* каталога исходных кодов Bitcoin Core.

В случае реализации вызова JSON-RPC в собственной программе для создания вызова можно использовать общую библиотеку HTTP, подобно тому, как это показано в предыдущем примере с curl.

Однако в большинстве популярных языков программирования существуют свои библиотеки, которые «оборачивают» API Bitcoin Core и значительно упрощают эту задачу. Для удобства доступа к API здесь будет использоваться библиотека python-bitcoinlib. Эта библиотека не является частью проекта Bitcoin Core, ее нужно установить обычным способом, как устанавливаются другие библиотеки Python.

Следует помнить, что для этого необходимо иметь работающий ресурс Bitcoin Core, который будет задействован для выполнения вызовов JSON-RPC.

Скрипт на Python в примере 3.4 выполняет простой вызов getblockchaininfo и распечатывает параметр блока из данных, возвращенных Bitcoin Core.

Пример 3.4 ❖ Запуск getblockchaininfo по протоколу JSON-RPC из API Bitcoin Core

```
from bitcoin.rpc import RawProxy

# Установление соединения с локальным узлом Bitcoin Core
p = RawProxy()

# Запуск команды getblockchaininfo  с сохранением результата выполнения в переменной info
info = p.getblockchaininfo()

# Извлечение элемента 'blocks' из переменной info
print(info['blocks'])
```

Запустив этот скрипт, можно получить следующий результат:

```
$ python rpc_example.py
773973
```

В нем сообщается, сколько блоков находится в блокчейне нашего локального узла Bitcoin Core. Не самый впечатляющий результат, но вполне наглядно показывающий основные возможности использования библиотеки в качестве упрощенного интерфейса JSON-RPC к API Bitcoin Core.

Далее при помощи вызовов `getrawtransaction` и `decodetransaction` можно получить информацию о платеже Алисы Бобу. В примере 3.5 получена транзакция Алисы и перечислены ее выходные данные. Для каждого выхода показан адрес получателя и сумма. Вспомним, что транзакция Алисы имела один выход для оплаты Бобу и один выход для возврата сдачи Алисе.

Пример 3.5 ❖ Получение транзакции и итерация ее выходов

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# ID транзакции Алисы
txid = "4662003086962l5bbc949d5141a49a4138ecdfdfaa2a8029c1f9bcecd1f96177"

# Сначала извлекается сырая транзакция в шестнадцатеричном формате
raw_tx = p.getrawtransaction(txid)

# Преобразование шестнадцатеричного формата транзакции в объект JSON
decoded_tx = p.decoderawtransaction(raw_tx)

# Получение всех выходных данных из этой транзакции
for output in decoded_tx['vout']:
    print(output['scriptPubKey']['address'], output['value'])
```

Запустив этот скрипт, можно получить следующий результат:

```
$ python rpc_transaction.py
bc1p8dqa4wjvnt890qmfws83te0v3qxzsfu7ul63kp7u56w8qc0qwp5qv995qn 0.00020000
bc1qwafvze0200nh9vkq4jmlf4sy0tn0ga5w0zpkpg 0.00075000
```

Оба предыдущих примера довольно просты. Для их выполнения не нужна программа; с тем же успехом можно использовать `bitcoin-cli`. Но следующий пример предполагает несколько сотен вызовов RPC и более наглядно демонстрирует использование программного интерфейса.

В примере 3.6 сначала извлекается блок, потом из него извлекаются все транзакции по ссылке на идентификатор каждой транзакции. Затем выполняется итерационный перебор всех выходов транзакции и суммирование полученных значений.

Пример 3.6 ❖ Получение блока и суммирование всех выходов транзакции

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# Высота блока, в котором записана транзакция Алисы
blockheight = 775072

# Получение хеш-значения блока с указанной высотой
blockhash = p.getblockhash(blockheight)
```

```
# Извлечение блока по его хешу
block = p.getblock(blockhash)

# Элемент tx содержит список идентификаторов всех транзакций этого блока

transactions = block['tx']

block_value = 0

# Итеративный проход по всем идентификаторам транзакций в блоке
for txid in transactions:
    tx_value = 0
    # Извлечение сырых данных транзакции по ее идентификатору
    raw_tx = p.getrawtransaction(txid)
    # Расшифровка транзакции
    decoded_tx = p.decoderawtransaction(raw_tx)
    # Итеративный проход по всем выходным данным транзакции
    for output in decoded_tx['vout']:
        # Добавление значений каждого выхода к общей сумме транзакции
        tx_value = tx_value + output['value']

    # Добавление суммы транзакции к общей сумме блока
    block_value = block_value + tx_value

print("Total value in block: ", block_value)
```

Запустив этот скрипт, можно получить следующий результат:

```
$ python rpc_block.py

Total value in block: 10322.07722534
```

По расчетам в этом примере общая сумма транзакций в данном блоке составляет 10 322,07722534 BTC (включая сумму оплаты в 25 BTC и комиссию в 0,0909 BTC). Теперь можно сравнить эту сумму с той, которую выдает проводник блокчейна при поиске хеша или высоты блока. Некоторые блокчейн-обработчики сообщают общую сумму без указания оплаты и комиссии. Проверьте, сможете ли вы обнаружить разницу.

Альтернативные клиенты, библиотеки и инструментарии

В экосистеме Биткойн есть множество альтернативных клиентов, библиотек, наборов инструментов и даже реализаций полноценных узлов. Они выполнены на различных языках программирования, что позволяет создать собственные интерфейсы на удобном для программистов языке.

В следующих разделах перечислены некоторые из лучших библиотек, клиентов и наборов инструментов. Они сгруппированы по языкам программирования.

C/C++

Bitcoin Core (https://github.com/bitcoin/bitcoin)
Эталонная реализация Биткойна.

JavaScript

bcoin (https://bcoin.io/)
Модульная и масштабируемая полноузловая реализация с API.

Bitcore (https://bitcore.io/)
Полная реализация узла, API и библиотека от Bitpay.

BitcoinJS (https://github.com/bitcoinjs/bitcoinjs-lib)
Библиотека Bitcoin на чистом JavaScript для node.js и браузеров.

Java

bitcoinj (https://bitcoinj.github.io/)
Библиотека клиента полноценного узла на Java.

Python

python-bitcoinlib (https://github.com/petertodd/python-bitcoinlib)
Библиотека для работы с биткойнами, библиотека консенсуса и узла на языке Python, разработанная Питером Тоддом (Peter Todd).

pycoin (https://github.com/richardkiss/pycoin)
Библиотека для работы с биткойнами на языке Python от Ричарда Кисса (Richard Kiss).

Go

btcd (https://github.com/btcsuite/btcd)
Полноценный биткойн-клиент на языке Go.

Rust

rust-bitcoin (https://github.com/rust-bitcoin/rust-bitcoin)
Библиотека Rust для сериализации, парсинга и вызовов API биткойнов.

Scala

bitcoin-s (https://bitcoin-s.org/)
Реализация Биткойна на языке Scala.

C#

NBitcoin (https://github.com/MetacoSA/NBitcoin)
Исчерпывающая библиотека для работы с биткойнами на платформе .NET.

Множество библиотек доступно и на других языках программирования, и их количество постоянно увеличивается.

Если вы следовали инструкциям в этой главе, теперь у вас работает Bitcoin Core, и вы начали изучать сеть и блокчейн с помощью собственного полноценного узла. Отныне вы можете самостоятельно использовать контролируемое вами программное обеспечение на контролируемом вами компьютере для проверки соответствия полученных вами биткойнов всем правилам системы Биткойн, не полагаясь при этом на сторонние инстанции. В следующих главах книги рассказывается о правилах системы и о том, как ваш узел и ваш кошелек используют их для защиты ваших средств, сохранения конфиденциальности и удобного расхода и приема.

Глава 4

Ключи и адреса

Алиса намерена заплатить Бобу, но тысячи полноценных узлов Биткойна, которые будут проверять ее транзакцию, не узнают, кто такие Алиса или Боб, и лучше оставить все как есть для защиты их конфиденциальности. Алисе нужно передать сообщение о передаче Бобу некоторого количества ее биткойнов, не связывая ни один аспект этой транзакции с реальной личностью Боба или с другими биткойн-платежами, которые получает Боб. Используемый Алисой метод должен гарантировать, что только Боб сможет в дальнейшем потратить полученные им биткойны.

В первоначальном варианте публикации о Биткойне была описана очень простая схема для достижения этих целей. Она показана на рис. 4.1.

Рис. 4.1 ❖ Цепочка транзакций из исходной публикации о Биткойне

Получатель средств, например Боб, использует открытый ключ (public key) для приема биткойнов в транзакции за подписью отправителя (например, Алисы). Расходуемые Алисой биткойны были ранее получены на один из ее открытых ключей, и она использует соответствующий секретный ключ (private

key) для создания своей подписи. Полноценные узлы могут удостовериться, что подпись Алисы соответствует выходу хеш-функции, которая также соответствует открытому ключу Боба и другим параметрам транзакции.

В этой главе будут рассмотрены открытые ключи, секретные ключи, подписи и хеш-функции. Далее все они будут использоваться вместе для описания адресов, которые применяются в современном программном обеспечении Биткойна.

Криптография с открытым ключом

Криптография с открытым ключом (public key cryptography), изобретенная в 1970-х годах, является математической основой современной компьютерной и информационной безопасности.

С момента изобретения криптографии с открытым ключом было создано несколько соответствующих математических функций, таких как возведение в степень простых чисел и умножение на эллиптических кривых. Эти математические функции достаточно просто просчитать в одном направлении и практически невозможно в обратном, по крайней мере с помощью доступных сегодня компьютеров и алгоритмов. Криптография, основанная на таких математических функциях, позволяет создавать цифровые подписи, не поддающиеся подделке. В качестве основы криптографии в системе Биткойн используются сложение и умножение на эллиптических кривых.

В Биткойне можно использовать криптографию с открытым ключом для создания ключевых пар, управляющих доступом к биткойнам. Пара ключей состоит из секретного и открытого ключей, образованных из секретного ключа. Открытый ключ используется для получения средств, а секретный – для подписи транзакций по расходованию этих средств.

Между открытым и секретным ключами есть математическая взаимосвязь, которая позволяет использовать секретный ключ для подписи сообщений. Эти подписи могут быть проверены по открытому ключу без необходимости разглашения секретного ключа.

В некоторых реализациях кошельков для удобства секретный и открытый ключи хранятся вместе в виде пары ключей. Однако открытый ключ может быть вычислен по закрытому, поэтому также допускается хранение только секретного ключа.

В биткойн-кошельке хранится набор пар ключей, каждая из которых состоит из секретного и открытого ключей. Секретный ключ (k) – это число, обычно определяемое случайным образом. Для получения открытого ключа (K) из закрытого ключа используется односторонняя криптографическая функция умножения на эллиптических кривых.

Почему асимметричная криптография (открытые/секретные ключи)?

Почему в системе Биткойн используется асимметричная криптография? Совсем не для «шифрования» (засекречивания) транзакций. Очень полезным свойством асимметричной криптографии является способность генерировать цифровые

подписи. Секретный ключ можно использовать для создания цифровой подписи под транзакцией. Такую подпись может создать только обладатель секретного ключа. В то же время любой пользователь с доступом к открытому ключу и транзакции может проверить подлинность подписи. Это полезное свойство асимметричной криптографии дает возможность всем желающим проверять каждую подпись под каждой транзакцией. При этом только владельцы секретных ключей могут создавать подлинные подписи.

Секретные ключи

Секретный ключ – это просто случайно выбранное число. Доступ к секретному ключу является основой контроля пользователя над всеми средствами, привязанными к соответствующему открытому ключу Bitcoin. С помощью секретного ключа можно создавать подписи для расходования биткойнов, подтверждая тем самым контроль над денежными средствами при совершении транзакции. Секретный ключ должен всегда храниться в тайне, поскольку его раскрытие посторонним равносильно передаче им контроля над биткойнами, защищенными этим ключом. Секретный ключ также следует хранить в резервной копии и защищать от случайной потери, поскольку в случае его пропажи его невозможно восстановить, и защищенные им средства также будут утеряны навсегда.

Секретный ключ Биткойна – это просто число. Можно подобрать секретный ключ случайным образом с помощью монеты, карандаша и бумаги: нужно подбросить монету 256 раз – и в результате получить двоичные цифры случайного секретного ключа, который можно использовать в биткойн-кошельке. Открытый ключ может быть сгенерирован на основе секретного ключа. Однако нужно быть осторожным, поскольку любой процесс, который не является абсолютно случайным, может значительно снизить безопасность вашего секретного ключа и связанных с ним биткойнов.

Первый и наиболее важный шаг при создании ключей – поиск надежного источника произвольного (случайного) числа. Специалисты по компьютерной технике называют это «*энтропией*» (*entropy*). Создание ключа Биткойна практически равносильно задаче «выберите число от 1 до 2^{256}». Конкретный метод, используемый для выбора этого числа, не имеет значения, при условии отсутствия предсказуемости и повторяемости. В программном обеспечении Биткойна используются генераторы случайных чисел с криптографической защитой для получения 256-битной энтропии.

Если говорить точнее, секретный ключ может быть любым числом в диапазоне от 0 до $n - 1$ включительно, где n – это константа ($n = 1{,}1578 \times 10^{77}$, чуть меньше 2^{256}), определяемая как порядок эллиптической кривой, используемой в системе Биткойн (см. «Объяснение криптографии на эллиптических кривых»). Для создания такого ключа необходимо случайным образом выбрать 256-битное число и убедиться, что оно меньше n. В программировании это обычно достигается подачей более длинной строки случайных битов, полученных из источника произвольных чисел с криптографической безопасностью, в хеш-алгоритм SHA256, который позволяет удобным образом получить 256-битное

значение с возможностью интерпретации его в качестве числа. Если результат меньше n, секретный ключ найден. В противном случае нужно просто повторить попытку с другим случайным числом.

Не пишите собственный код для создания случайного числа и не используйте «простой» генератор случайных чисел, который предлагает ваш язык программирования. Используйте генератор псевдослучайных чисел с криптографической защитой (cryptographically secure pseudorandom number generator, CSPRNG) с исходным сигналом из источника с достаточной энтропией. Изучите документацию выбранной вами библиотеки генератора случайных чисел, чтобы убедиться в ее криптографической безопасности. Правильная реализация CSPRNG имеет критическое значение для безопасности ключей.

Ниже показан пример случайно сгенерированного секретного ключа (k) в шестнадцатеричном формате (256 бит представлены в виде 64 шестнадцатеричных цифр по 4 бита в каждой):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

Пространство секретных ключей биткойна (2^{256}) – непостижимо большое число. В десятичном исчислении это примерно 10^{77}. Для сравнения: видимая Вселенная, по некоторым оценкам, содержит 10^{80} атомов.

Объяснение криптографии на эллиптических кривых

Криптография на эллиптических кривых (Elliptic curve cryptography, ECC) – это тип асимметричной криптографии с открытым ключом, в основе которого заложена задача дискретного логарифма, выраженного через сложение и умножение в точках эллиптической кривой.

На рис. 4.2 приведен пример эллиптической кривой, аналогичной используемой в системе Биткойн.

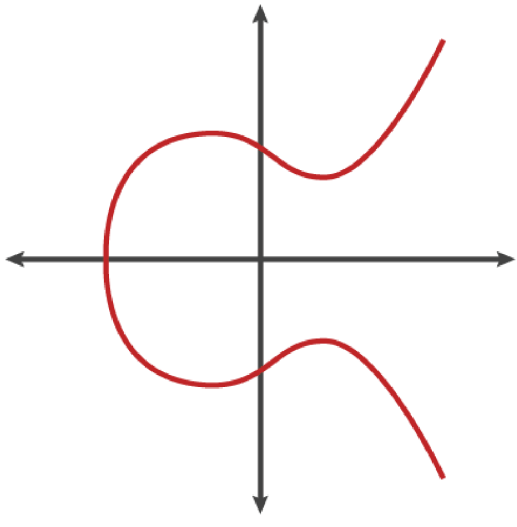


Рис. 4.2 ❖ Эллиптическая кривая

В Биткойне используется определенная эллиптическая кривая и набор математических констант по стандарту `secp256k1`, разработанному Национальным институтом стандартов и технологий США (National Institute of Standards and Technology, NIST). Согласно стандарту `secp256k1`, получить необходимую эллиптическую кривую можно с помощью следующей функции:

$y^2 = (x^3 + 7)$ над полем $(\mathbb{F}_p)$,

или

$y^2 \bmod p = (x^3 + 7) \bmod p$.

Выражение $\bmod p$ (modulo prime number p – остаток от деления на простое число p) указывает на то, что эта кривая находится над конечным полем простого числа p. Это число также обозначается как $\mathbb{F}_p$, где $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, то есть очень большое простое число.

Поскольку эта кривая определена над конечным полем простого числа, а не над полем действительных (вещественных) чисел, она выглядит как множество точек в двумерном пространстве, что усложняет ее визуализацию. Однако ее вычисление идентично математике эллиптической кривой над действительными числами. В качестве примера на рис. 4.3 показана та же эллиптическая кривая над гораздо меньшим конечным полем простого числа 17, изображенная в виде множества точек на координатной сетке. Эллиптическую кривую Биткойна по стандарту `secp256k1` можно представить в виде более сложной структуры из точек на неизмеримо большой координатной сетке.

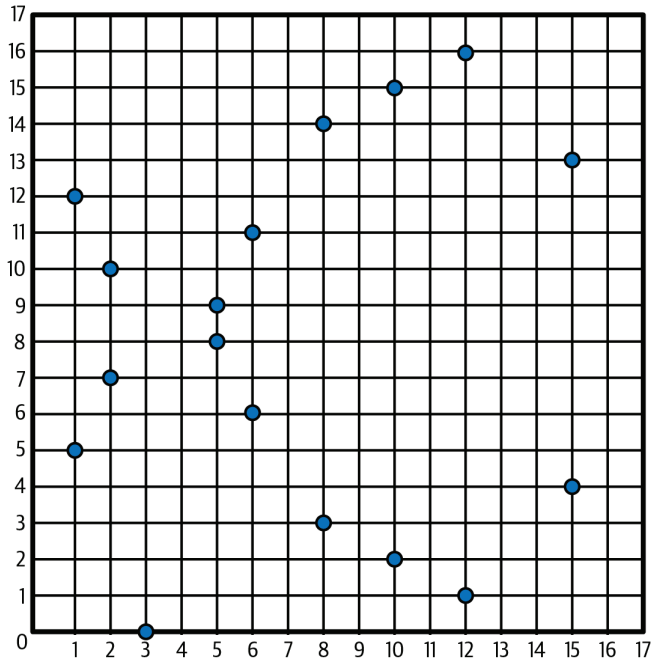


Рис. 4.3 ❖ Визуализация эллиптической кривой над полем $F(p)$, где $p = 17$

Так, например, точка P с координатами (x, y) является точкой на кривой по стандарту secp256k1:

```
P =
(55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)
```

В примере 4.1 показано, как можно проверить это самостоятельно с помощью языка Python.

Пример 4.1 ❖ Проверка расположения точки на эллиптической кривой с помощью Python

```
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
> p = 115792089237316195423570985008687907853269984665640564039457584007908834671663
> x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
> y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
> (x ** 3 + 7 - y**2) % p
0
```

В математике эллиптических кривых есть точка под названием «бесконечно удаленная точка» (point at infinity), которая примерно соответствует нулю в сложении. На компьютерах ее иногда представляют в виде x = y = 0. Это не удовлетворяет уравнению эллиптической кривой, но является простым частным случаем, который несложно проверить.

Также существует оператор +, называемый «оператором сложения» (addition). Его свойства схожи с традиционным сложением вещественных чисел, с которым дети знакомятся в начальной школе. Если на эллиптической кривой заданы две точки P_1 и P_2, существует третья точка $P_3 = P_1 + P_2$, также расположенная на эллиптической кривой.

Геометрически эту третью точку P_3 можно найти с помощью прямой между точками P^1 и P^2. Эта линия пересечет эллиптическую кривую в одном-единственном дополнительном месте. Назовем эту точку $P'_3 = (x, y)$. Отразив ее по оси x, получим $P_3 = (x, -y)$.

Есть несколько особых случаев, которые объясняют необходимость использования «бесконечно удаленной точки».

Если P_1 и P_2 являются одной и той же точкой, то линия «между» P_1 и P_2 будет касательной к кривой в этой точке P_1. Эта касательная пересечет кривую в одной-единственной точке. Для определения угла наклона касательной можно использовать методы математического анализа. Примечательно, что эти методы дают результат, даже если ограничиться точками на кривой с двумя целочисленными координатами!

В некоторых случаях (например, если P_1 и P_2 имеют одинаковые значения x, но разные значения y) касательная линия окажется строго вертикальной, и как раз в этом случае P_3 = «бесконечно удаленная точка».

Если P_1 является бесконечно удаленной точкой, тогда $P_1 + P_2 = P_2$. По аналогии если P_2 – бесконечно удаленная точка, тогда $P_1 + P_2 = P_1$. Таким образом, точка на бесконечности выступает в роли нуля.

Кроме того, операция сложения (+) является ассоциативной, то есть (A + B) + C = A + (B + C). Это значит, что можно писать A + B + C без скобок и без разночтений.

Теперь, когда дано определение операции сложения, можно определить умножение тем же стандартным способом. Пусть для точки P на эллиптической кривой k – целое число, тогда kP = P + P + P + … + P (k раз). Следует заметить, что k в таких случаях иногда ошибочно называют «экспонентой».

Открытые ключи

Открытый ключ вычисляется из секретного ключа с помощью эллиптического умножения кривых, которое является необратимым: $K = k \times G$, где k – секретный ключ, G – постоянная точка, называемая *точкой генерации* (*generator point*), а K – результирующий открытый ключ. Обратная операция, известная как «нахождение дискретного логарифма» (finding the discrete logarithm) – вычисление k по известному K, трудна настолько же, насколько сложна переборка всех возможных значений k (то есть поиск методом полного перебора).

Умножение на эллиптической кривой – это тип функции, которую специалисты по криптографии называют «функцией-ловушкой» (trap door): ее легко выполнить в одном направлении (умножение) и невозможно в обратном (деление). Обладатель секретного ключа может легко создать открытый ключ и затем поделиться им со всеми, будучи уверенным, что никто не сможет обратить эту функцию и вычислить секретный ключ из открытого. Эта математическая уловка лежит в основе надежных и безопасных цифровых подписей для подтверждения контроля над активами в биткойнах.

Имея секретный ключ в виде случайно сгенерированного числа k, необходимо умножить его на заранее определенную точку на кривой, которая называется *базовой точкой генерации G* (*generator point G*), чтобы получить другую точку в другом месте этой кривой. Именно она и является соответствующим открытым ключом K. Базовая точка генерации задается в рамках стандарта `secp256k1` и всегда остается неизменной для всех ключей биткойна:

$$K = k \times G,$$

где k – секретный ключ, G – точка генерации, а K – итоговый открытый ключ, точка на кривой. Поскольку точка генерации всегда одна и та же для всех пользователей системы Биткойн, секретный ключ k, умноженный на G, всегда будет выдавать один и тот же открытый ключ K. Связь между k и K неизменна, но может быть рассчитана только в одном направлении – от k к K. Именно поэтому открытый ключ Биткойна (K) может быть передан кому угодно и при этом не раскрывает секретный ключ пользователя (k).

Секретный ключ можно преобразовать в открытый, но открытый ключ нельзя преобразовать обратно в закрытый, потому что эта математика работает только в одном направлении.

Применяя умножение на эллиптической кривой, можно взять сгенерированный ранее секретный ключ k и умножить его на точку генерации G для получения открытого ключа K:

```
K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD × G
```

Открытый ключ K определяется как точка $K = (x, y)$:

$K = (x, y)$,

где

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Для наглядного геометрического представления умножения точки на целое число можно использовать более простую эллиптическую кривую над вещественными числами – как бы то ни было, математика та же самая. Задача состоит в поиске произведения kG на точку генерации G, что равносильно сложению G с самим собой k раз подряд. На эллиптических кривых добавление точки к самой себе равносильно проведению касательной линии к точке и нахождению места ее пересечения с кривой, а затем отображению этой точки на оси x.

На рис. 4.4 показан процесс вычисления G, $2G$, $4G$ в качестве геометрической операции на кривой.

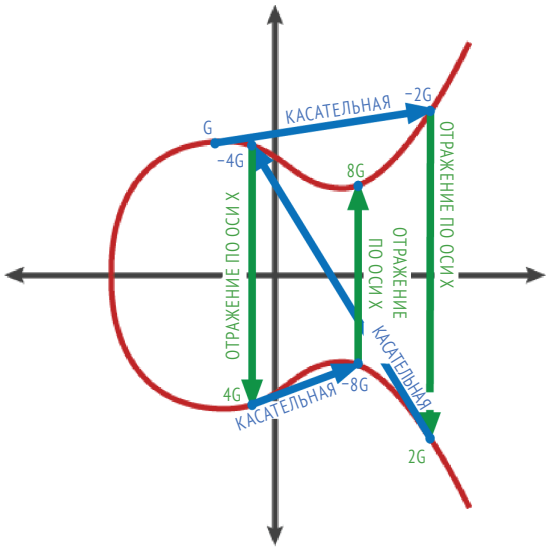


Рис. 4.4. Криптография на эллиптической кривой: визуализация умножения точки G на целое число k

Многие реализации Биткойна используют для вычислений на эллиптических кривых криптографическую библиотеку libsecp256k1 (https://github.com/bitcoin-core/secp256k1).

Скрипты выхода и входа

Несмотря на то что на рис. 4.1 из первоначальной публикации Биткойна показано применение открытых ключей (pubkeys) и подписей (sigs) напрямую,

в первой версии системы Биткойн платежи отправлялись в поле под названием «*скрипт выхода*» (*output script*), а отправка этих биткойнов санкционировалась полем под названием «*скрипт входа*» (*input script*). Эти поля дают возможность выполнять дополнительные операции вдобавок к (или вместо) проверке соответствия подписи открытому ключу. Например, скрипт выхода может содержать два открытых ключа и требовать наличия двух соответствующих подписей в скрипте ввода расходов.

Позже, в разделе «Скрипты транзакций и язык скриптов», будет более подробно рассказано о скриптах. Пока же достаточно помнить, что биткойны поступают в скрипт выхода, который действует как открытый ключ, а расходы биткойнов санкционируются скриптом входа, который действует как подпись.

IP-адреса: исходный адрес для Биткойна (P2PK)

Как было установлено ранее, Алиса может расплатиться с Бобом, передав часть своих биткойнов по одному из открытых ключей Боба. Но как ей получить один из этих открытых ключей? Боб мог бы просто передать ей копию, но теперь еще раз взглянем на открытый ключ, рассмотренный в разделе «Открытые ключи». Он довольно длинный. Только представьте, как Боб пытается надиктовать его Алисе по телефону:

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

В ранних версиях программного обеспечения Биткойна вместо прямого ввода открытого ключа можно было ввести IP-адрес получателя, как показано на рис. 4.5 (взято из The Internet Archive, https://web.archive.org/web/20090722011820/https://bitcoin.org/). Позже эта функция была удалена из-за множества проблем с использованием IP-адресов, однако ее краткое описание может помочь лучше понять смысл добавления некоторых функций в протокол Биткойна.

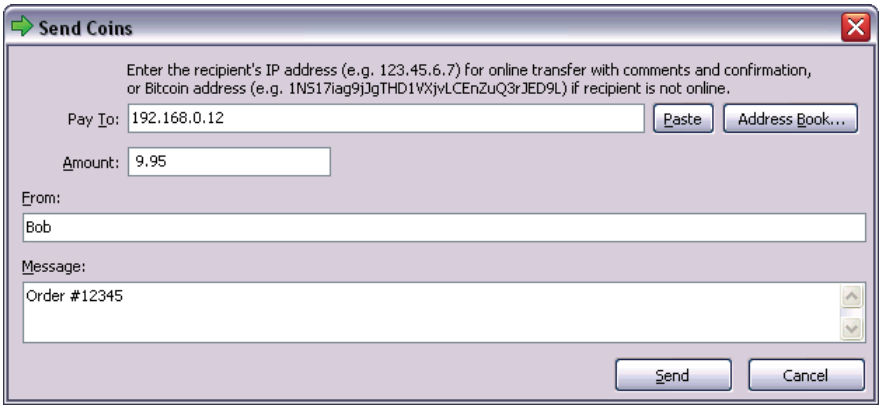


Рис. 4.5 ❖ Ранняя версия экрана отправки для Биткойна

Если Алиса вводила IP-адрес Боба в системе Bitcoin 0.1, ее полный узел устанавливал соединение с его полным узлом и получал новый открытый ключ из кошелька Боба. Ранее его узел никому этот ключ не передавал. Этот новый открытый ключ был важен для исключения вероятности сопоставления воедино различных транзакций с оплатой Бобу, как если бы кто-то, просматривая блокчейн, заметил, что все транзакции оплачиваются одним и тем же открытым ключом.

Используя открытый ключ, полученный ее узлом от узла Боба, кошелек Алисы создавал выход транзакции для оплаты с помощью очень простого скрипта выхода:

```
<Bob's public key> OP_CHECKSIG
```

Позже Боб сможет использовать этот выход со скриптом входа, полностью состоящим из его подписи:

```
<Bob's signature>
```

Для понимания работы выходного и входного скриптов можно объединить их вместе (сначала идет скрипт входа), а затем принять во внимание, что каждый фрагмент данных (в угловых скобках) помещается в начало списка элементов, который называется стеком (stack). При появлении кода операции (opcode) используются элементы из стека, начиная с самых верхних. Для начала можно проследить, как это работает в комбинированном скрипте:

```
<Bob's signature> <Bob's public key> OP_CHECKSIG
```

В этом скрипте подпись Боба помещается в стек, затем поверх нее располагается открытый ключ Боба. Команда `OP_CHECKSIG` требует двух элементов, сначала открытый ключ, а затем подпись, и удаляет их из стека. Она проверяет соответствие подписи открытому ключу, а также подтверждает (подписывает) различные поля транзакции. Если подпись верна, `OP_CHECKSIG` заменяет себя в стеке на значение 1; если подпись не верна, то на 0. Если на момент завершения оценки на вершине стека остается ненулевой элемент, скрипт работает. Если все скрипты в транзакции прошли и все остальные детали транзакции достоверны, то все полные узлы будут считать транзакцию достоверной.

Словом, в предыдущем сценарии используются тот же открытый ключ и подпись, что описаны в оригинальной статье, но с дополнительными сложностями в виде двух полей скрипта и операционного кода. Это кажется лишним делом, но преимущества этого можно будет увидеть в следующем разделе.

Данный тип выхода известен сегодня как «*оплата открытым ключом*» (*pay to public key*), или сокращенно P2PK. Он никогда широко не использовался для платежей, и ни одна распространенная программа не поддерживает платежи по IP-адресам уже почти десятилетие.

Устаревшие адреса для P2PKH

Использование IP-адреса пользователя, которому вы собираетесь заплатить, обладает как плюсами, так и минусами. В частности, одним из минусов можно

назвать необходимость наличия у получателя онлайн-кошелька с IP-адресом, доступ к которому открыт постоянно из любого места. Для многих такой вариант неприемлем. Они выключают свои компьютеры на ночь, их ноутбуки уходят в спящий режим, находятся за межсетевым экраном или используют трансляцию сетевых адресов (Network Address Translation, NAT).

Это возвращает нас к вопросу о необходимости передавать длинный открытый ключ получателям, таким как Боб, и расходующим средства, таким как Алиса. Самая короткая версия открытых ключей Биткойна, известная разработчикам раннего протокола системы, составляла 65 байт, что соответствует 130 символам при записи в шестнадцатеричном формате. Вместе с тем в Биткойне уже содержится множество структур данных размером гораздо больше 65 байт. Для безопасной ссылки на них в других частях Биткойна необходимо использовать наименьший объем данных, обеспечивающий их безопасность.

Для этого в Биткойне используется *хеш-функция* (*hash function*) – то есть функция, которая получает изначально большой объем данных, скремблирует их (хеширует) и выдает на выходе фиксированный объем данных. Криптографическая хеш-функция всегда выдает один и тот же результат при одинаковом входном сигнале, а защищенная функция также не позволяет кому-либо подобрать другие входные данные для получения ранее показанного результата. Таким образом, выход представляет собой *обязательство* (*commitment*) по отношению к входу. Обещание, что на практике только данные на входе *x* будут выдавать результат *X*.

Например, представьте, что вам хотят задать вопрос и при этом предоставить свой вариант ответа в форме, которую вы не сможете сразу же прочитать. Допустим, вопрос звучит так: «в каком году Сатоши Накамото начал работу над Биткойном?» Ответ на вопрос будет представлен вам в виде выхода хеш-функции SHA256 – функции, наиболее часто используемой в системе Биткойн:

```
94d7a772612c8f2f2ec609d41f5bd3d04a5aa1dfe3582f04af517d396a302e4e
```

Позже, когда вы сообщите свой ответ на вопрос, можно будет раскрыть предложенный ответ и доказать вам, что этот ответ при вводе в хеш-функцию дает точно такой же результат, как и предложенный вами ранее:

```
$ echo "2007.  He said about a year and a half before Oct 2008" | sha256sum
94d7a772612c8f2f2ec609d41f5bd3d04a5aa1dfe3582f04af517d396a302e4e
```

Теперь представьте, что к Бобу обратились с вопросом: «Какой у вас открытый ключ?» Боб может использовать хеш-функцию для передачи нам криптографически защищенного обязательства по своему открытому ключу. Если позже он раскроет свой ключ, и при проверке он выдаст то же самое обязательство, что и ранее, можно не сомневаться, что это был точно такой же ключ, как и при создании предыдущего обязательства.

Хеш-функция SHA256 считается очень безопасной и выдает на выходе 256 бит (32 байта), что менее половины размера оригинальных открытых ключей Биткойна. Однако существуют и другие, чуть менее безопасные хеш-функции, которые выдают на выходе меньший размер, например хеш-функция RIPEMD-160, чей выход составляет 160 бит (20 байт). По причинам, которые

Сатоши Накамото так и не озвучил, в первоначальной версии Биткойна обязательства по открытым ключам сначала хешировались с помощью SHA256, а затем с помощью RIPEMD-160; в результате обязательства по открытому ключу имели размер 20 байт.

Рассмотрим этот алгоритм. Сначала из открытого ключа *K* вычисляется хеш SHA256, а затем результат хешируется RIPEMD-160, в результате чего получается 160-битное (20-байтовое) число:

$$A = \text{RIPEMD160}(\text{SHA256}(K)),$$

где *K* – открытый ключ, *A* – полученное обязательство.

Теперь, когда понятно, как сделать обязательство для открытого ключа, необходимо выяснить, как использовать его в транзакции. Рассмотрим следующий скрипт выхода:

```
OP_DUP OP_HASH160 <обязательство Боба> OP_EQUAL OP_CHECKSIG
```

И также следующий входной скрипт:

```
<подпись Боба> <открытый ключ Боба>.
```

Вместе они образуют следующий скрипт:

```
<sig> <pubkey> OP_DUP OP_HASH160 <commitment> OP_EQUALVERIFY OP_CHECKSIG
```

Как и в разделе «IP-адреса: исходный адрес для Биткойна (P2PK)», выкладываем элементы в стек. Сначала идет подпись Боба, затем его открытый ключ помещается на вершину стека. Команда `OP_DUP` дублирует верхний элемент, так что теперь верхний и предпоследний элементы в стеке оба являются открытыми ключами Боба. Команда `OP_HASH160` поглощает (удаляет) верхний открытый ключ и заменяет его результатом хеширования `RIPEMD160(SHA256(K))`, так что теперь вершиной стека является хеш открытого ключа Боба. Затем в вершину стека добавляется обязательство открытого ключа Боба. Команда `OP_EQUALVERIFY` использует два верхних элемента и убеждается в их равенстве. Это действительно должно быть так, если открытый ключ, предоставленный Бобом во входном скрипте, является тем же открытым ключом, который использовался для создания обязательства в выходном скрипте, оплаченном Алисой. Если `OP_EQUALVERIFY` не выполняется, весь скрипт тоже не выполняется. Наконец, остается стек, содержащий только подпись Боба и его открытый ключ. Команда `OP_CHECKSIG` проверяет их на соответствие друг другу и что подпись подтверждает транзакцию.

Несмотря на то что процесс оплаты хеша открытого ключа (pay to public key hash, P2PKH) может показаться запутанным, он дает Алисе возможность передать Бобу только обязательство в 20 байт по его открытому ключу вместо самого ключа, который в первоначальной версии Биткойна был бы равен 65 байтам. Это гораздо меньший объем данных, который Боб должен передать Алисе.

Однако пока не обсуждался вопрос о том, как Боб передает эти 20 байт из своего биткойн-кошелька в кошелек Алисы. Существуют общепринятые кодировки значений байтов, в том числе шестнадцатеричная, но любая ошибка при копировании обязательства приведет к отправке биткойнов на нерасходуемый

выход, в результате чего они будут навсегда утеряны. В следующем разделе будут рассмотрены вопросы компактного кодирования и надежных контрольных сумм.

Кодирование Base58check

Для компактного представления длинных чисел с использованием меньшего количества символов во многих компьютерных системах применяются смешанные алфавитно-цифровые представления с основанием (base, radix) более 10. Так, если в традиционной десятичной системе используется 10 цифр, от 0 до 9, то в шестнадцатеричной – 16, а в качестве шести дополнительных символов используются буквы от A до F. Число, представленное в шестнадцатеричном формате, короче, чем равнозначное в десятичном. Еще более компактное представление формата base64 использует 26 строчных букв, 26 прописных букв, 10 цифр и еще 2 символа, такие как «+» и «/», для передачи двоичных данных с помощью текстовых средств, таких как электронная почта.

Кодировка base58 похожа на base64, в ней используются прописные и строчные буквы, цифры, но отсутствуют некоторые символы, которые часто путают друг с другом и которые могут выглядеть идентичными при отображении некоторыми шрифтами. В частности, base58 – это base64 без 0 (нуля), O (заглавной буквы o), l (строчной буквы L), I (заглавной буквы i), а также символов «+» и «/». Или, проще говоря, это набор строчных и прописных букв и цифр без четырех (0, O, l, I), которые были упомянуты выше. В примере 4.2 показан полный набор символов алфавита base58.

Пример 4.2 ❖ Алфавит Биткойна base58

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

Для дополнительной защиты от опечаток и ошибок транскрипции в состав base58check включена *контрольная сумма* (*checksum*), закодированная в алфавите base58. Контрольная сумма – это четыре дополнительных байта, добавляемых в конец кодируемых данных. Контрольная сумма рассчитывается на основе хеша закодированных данных и поэтому может использоваться для выявления ошибок при расшифровке и наборе текста. При получении кода base58check программа декодирования вычисляет контрольную сумму данных и сравнивает ее с контрольной суммой в коде. Если они не совпадают, произошла ошибка, и данные base58check недействительны. Это позволяет избежать ошибочного ввода адреса Биткойна, который программа кошелька может принять за действительное направление. Такая ошибка может привести к потере средств.

Чтобы преобразовать данные (число) в формат base58check, сначала нужно добавить к ним префикс, называемый «байтом версии» (version byte). Он позволяет легко определить тип закодированных данных. Например, префикс zero (0x00 в шестнадцатеричном формате) указывает на необходимость использования данных в качестве обязательства (хеша) в устаревшем скрипте выхода P2PKH. Список распространенных префиксов версий приведен в табл. 4.1.

Таблица 4.1. Примеры префикса версии Base58check и закодированного результата

Тип	Префикс версии (16-ричный)	Префикс результата Base58
Адрес для оплаты по хешу открытого ключа (P2PKH)	0x00	1
Адрес для оплаты по хешу скрипта (P2SH)	0x05	3
Адрес для тестирования в Testnet, P2PKH	0x6F	m или n
Адрес для тестирования в Testnet, P2SH	0xC4	2
Секретный ключ WIF	0x80	5, K или L
Расширенный открытый ключ по BIP32	0x0488B21E	xpub

Далее мы вычисляем контрольную сумму «double-SHA», то есть предыдущий результат (префикс, скомбинированный с данными) дважды обрабатывается алгоритмом хеширования SHA256:

```
checksum = SHA256(SHA256(prefix||data))
```

Из полученного 32-байтового хеша («хеш из хеша») нужно взять только первые четыре байта. Эти четыре байта служат кодом для проверки на наличие ошибок, или контрольной суммой (checksum). Контрольная сумма добавляется в конец.

Результат состоит из трех элементов: префикса, данных и контрольной суммы. Результат кодируется с помощью алфавита base58, описанного выше. На рис. 4.6 показан процесс кодирования base58check: формат base58, версионированный и дополненный контрольной суммой для безошибочного кодирования данных биткойна.

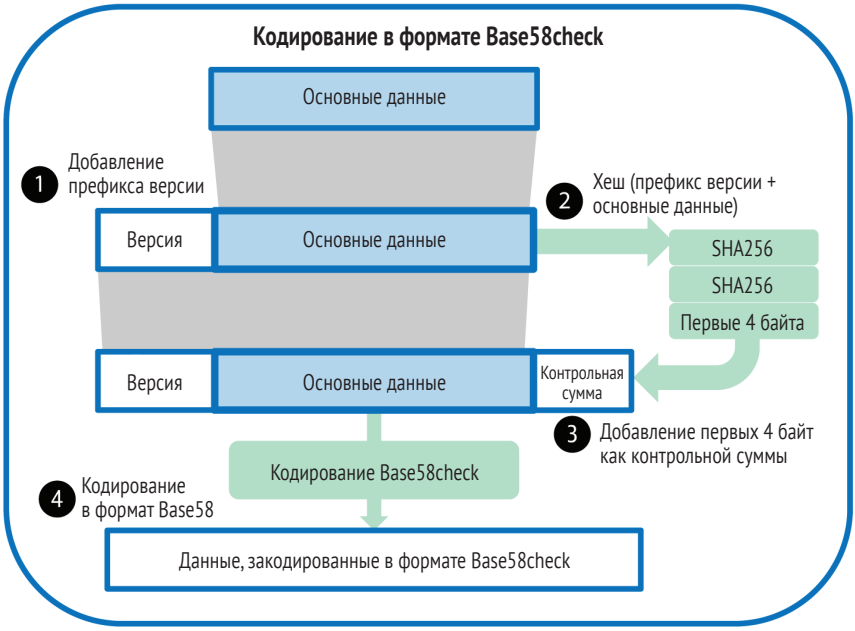


Рис. 4.6 ❖ Кодирование с помощью base58check

Кроме обязательств по открытому ключу, другие данные в Биткойне представляются пользователю в кодировке base58check для обеспечения компактности, удобства чтения и выявления ошибок. Префикс версии в кодировке base58check используется для получения легко различимых форматов, которые в кодировке base58 включают специальные символы в начале кодированной в base58check полезной нагрузки. Эти символы упрощают идентификацию типа закодированных данных и их использование. Именно это отличает, например, закодированный в base58check адрес Биткойна, начинающийся с 1, от закодированного в base58check секретного ключа в формате импорта кошелька (Wallet Import Format, WIF), начинающегося с 5. Некоторые примеры префиксов версий и соответствующих символов base58 были приведены в табл. 4.1.

На рис. 4.7 показана комбинация открытых ключей, обязательств на основе хеша и кодировки base58check, которая наглядно отражает преобразование открытого ключа в адрес Биткойна.

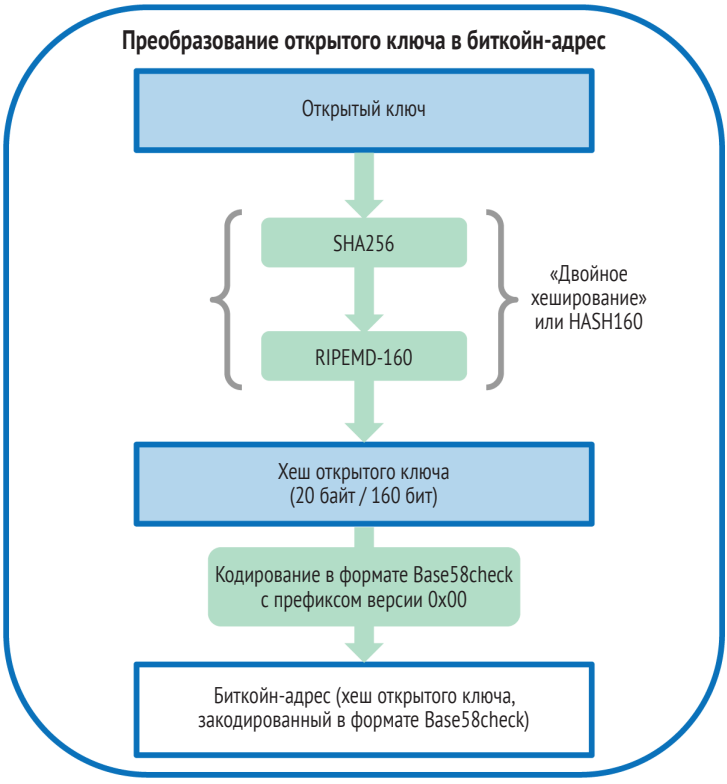


Рис. 4.7 ❖ Преобразование открытого ключа в адрес Биткойна

Сжатые открытые ключи

Во времена разработки Биткойна его создатели имели возможность создавать только 65-байтовые открытые ключи. Однако позже один из разработчи-

ков познакомился с альтернативной кодировкой открытых ключей, которая использовала всего 33 байта и обладала обратной совместимостью со всеми полноценными узлами Биткойна того времени, так что не было необходимости в изменении протокола Биткойна. Эти 33-байтовые открытые ключи называют *сжатыми открытыми ключами* (*compressed public keys*), а первоначальные 65-байтовые ключи – *несжатыми открытыми ключами* (*uncompressed public keys*). Использование более компактных открытых ключей позволяет сократить объем транзакций, что дает возможность проводить больше платежей в одном блоке.

Как уже было сказано в разделе «Открытые ключи», открытый ключ – это точка (x, y) на эллиптической кривой. Поскольку кривая является отображением математической функции, точка на кривой представляет собой решение уравнения, и поэтому, если известна координата *x*, можно вычислить координату *y*, решив уравнение $y^2 \bmod p = (x^3 + 7) \bmod p$. Это позволяет хранить только координату точки *x* открытого ключа, опустив координату *y* и сократив размер ключа и места, необходимого для его хранения, на 256 бит. Сокращение размера почти на 50 % при каждой транзакции дает со временем значительную экономию!

Ниже приведен открытый ключ, сгенерированный на основе секретного ключа из раздела «Открытые ключи»:

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Ниже представлен тот же открытый ключ в виде 520-битного числа (130 шестнадцатеричных цифр) с префиксом `04`, за которым следуют координаты *x*, а затем *y*, в виде `04` x y:

```
K = 04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A\
    07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Если несжатые открытые ключи имеют префикс `04`, то сжатые открытые ключи начинаются с префикса `02` или `03`. Теперь разберемся, почему возможны два префикса: поскольку левая часть уравнения равна y^2, решением для *y* является квадратный корень, который может иметь положительное или отрицательное значение. Визуально это выражается в виде того, что полученная координата *y* может находиться выше или ниже оси *x*. Как видно из графика эллиптической кривой на рис. 4.2, кривая симметрична, то есть отражается от оси *x* как в зеркале. И хотя можно не указывать координату *y*, необходимо сохранить ее *знак* (положительный или отрицательный). Другими словами, нужно помнить, была ли она выше или ниже оси *x*, поскольку каждый из этих вариантов представляет собой отдельную точку и отдельный открытый ключ. При вычислении на эллиптической кривой в бинарной арифметике на конечном поле простого порядка *p* координата *y* является либо четной, либо нечетной, что соответствует знаку плюс/минус, как уже говорилось ранее. Поэтому чтобы различать два возможных значения *y*, сжатый открытый ключ хранится с префиксом `02`, если координата *y* четная, и `03`, если нечетная, что позволяет программе корректно вывести координату *y* из координаты *x* и распаковать открытый ключ до получения полных координат точки. Сжатие открытого ключа показано на рис. 4.8.

Рис. 4.8 ❖ Сжатие открытого ключа

Ниже представлен тот же открытый ключ из раздела «Открытые ключи» в виде сжатого открытого ключа, хранящегося в 264 битах (66 шестнадцатеричных цифр) с префиксом `03`, указывающим на нечетность координаты *y*:

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Этот сжатый открытый ключ соответствует тому же закрытому ключу, то есть он создан на основе того же секретного ключа. Однако выглядит он не так, как несжатый открытый ключ. Более того, если преобразовать этот сжатый открытый ключ в обязательство с помощью функции HASH160 (`RIPEMD160(SHA256(K))`), то получится *другое* обязательство, отличное от обязательства для несжатого открытого ключа, что приведет к появлению другого адреса. Это приводит к путанице, поскольку подразумевает, что один секретный ключ может создавать открытый ключ в двух разных форматах (сжатом и несжатом), которые дают два разных адреса Биткойна, хотя секретный ключ идентичен для обоих адресов.

В настоящее время сжатые открытые ключи используются по умолчанию почти во всех программах для работы с Биткойном. Они необходимы при использовании некоторых новых функций, добавленных в последующих обновлениях протокола.

Однако некоторым программам все еще требуется поддержка несжатых открытых ключей. Таким как, например, приложение кошелька с поддержкой импорта секретных ключей из устаревшего кошелька. Когда новый кошелек ска-

нирует блокчейн на предмет старых выходов и входов P2PKH, ему необходимо определиться, сканировать 65-байтовые ключи (и обязательства по ним) или 33-байтовые ключи (и обязательства по ним). Отсутствие возможности просканировать правильный вариант может привести к потере возможности использовать весь свой баланс. Для решения этой проблемы при экспорте секретных ключей из кошелька формат импорта WIF, используемый для их представления, в новых кошельках Биткойна реализован несколько иначе, указывая на использование этих секретных ключей для создания сжатых открытых ключей.

Устаревший скрипт Pay to Script Hash (P2SH)

Как уже говорилось в предыдущих разделах, получатель биткойнов (например, Боб) может потребовать, чтобы платежи в его адрес содержали определенные ограничения в скрипте выхода. При этом Бобу необходимо соблюсти эти ограничения с помощью входного скрипта, когда он будет расходовать эти биткойны. В разделе «IP-адреса: оригинальный адрес для биткойна (P2PK)» ограничение сводилось к тому, что входной скрипт должен содержать соответствующую подпись. В разделе «Устаревшие адреса для P2PKH» также требовалось представить соответствующий открытый ключ.

Чтобы покупатель (например, Алиса) мог внести требуемые Бобом ограничения в скрипт выхода, который она использует для оплаты, Боб должен сообщить ей об этих ограничениях. Это напоминает проблему, когда Бобу нужно сообщить ей свой открытый ключ. Подобно тому, как в этом случае открытые ключи могут быть довольно большими, используемые Бобом ограничения также могут иметь большой размер – потенциально тысячи байт. Это не только тысячи байт для передачи Алисе, но и тысячи байт, за которые ей придется платить комиссионные за транзакции каждый раз, когда она захочет отправить деньги Бобу. Однако и здесь можно использовать хеш-функции для создания небольших обязательств к большим объемам данных.

Обновление BIP16 для протокола Биткойна, выпущенное в 2012 году, позволяет выходному скрипту принимать обязательства перед «*скриптом погашения*» (*redemption script*, *redeem script*). Когда Боб тратит свои биткойны, его входной скрипт должен предоставить скрипт погашения, соответствующий обязательству, а также любые данные, необходимые для удовлетворения этого скрипта погашения (скажем, подписи). Например, Боб хочет получить две подписи, чтобы потратить свои биткойны: одну подпись из своего кошелька на настольном компьютере и одну из аппаратного устройства для подписи. Он вводит эти условия в скрипт погашения:

```
<public key 1> OP_CHECKSIGVERIFY <public key 2> OP_CHECKSIG
```

Затем он создает обязательство для скрипта погашения с помощью того же механизма HASH160, который используется для обязательств P2PKH, `RIPEMD160(SHA256(script))`. Это обязательство вводится в выходной скрипт посредством специального шаблона:

```
OP_HASH160 <commitment> OP_EQUAL
```

При применении Pay to Script Hash (P2SH) нужно использовать специальный шаблон P2SH без дополнительных данных или условий в выходном скрипте. Если выходной скрипт не будет точно соответствовать OP_HASH160 <20 байт> OP_EQUAL, скрипт погашения не сможет быть использован, и все биткойны могут оказаться либо неизрасходованными, либо потраченными кем угодно (то есть их может взять кто угодно).

Когда Боб собирается потратить платеж, полученный им на обязательства по своему скрипту, он использует входной скрипт, включающий скрипт погашения, который в этом случае оформляется в виде одного сериализованного элемента данных. Он также предоставляет подписи, необходимые для выполнения скрипта погашения, располагая их в том порядке, в котором они будут использоваться операционными кодами:

```
<signature2> <signature1> <redeem script>
```

После получения от Боба его средств полные узлы Биткойна удостоверяются, что сериализованный скрипт погашения хеширует то же значение, что и обязательство. Затем они заменят его в стеке на его десериализованное значение:

```
<signature2> <signature1> <pubkey1> OP_CHECKSIGVERIFY <pubkey2> OP_CHECKSIG
```

Выполняется скрипт, и если он проходит, а все остальные детали транзакции верны, транзакция считается действительной.

Адреса для P2SH также создаются с помощью алфавита base58check. Префикс версии равен 5, в результате чего кодированный адрес начинается с 3. Один из примеров адреса P2SH – `3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM`.

P2SH – это совсем не обязательно транзакция с несколькими подписями. Чаще всего адрес P2SH действительно представляет собой скрипт со множеством подписей, но он также может быть скриптом для кодирования других типов транзакций.

P2PKH и P2SH являются единственными двумя шаблонами скриптов, которые использовались с кодировкой base58check. Сейчас они известны как устаревшие адреса (legacy addresses) и со временем встречаются все реже. Устаревшие адреса вытесняются семейством адресов bech32.

Коллизионные атаки на P2SH

Все адреса на основе хеш-функций теоретически уязвимы к тому, что злоумышленник может самостоятельно найти тот же вход, который обеспечил выход хеш-функции (обязательство). В случае с Биткойном, если злоумышленник найдет входные данные так же, как это сделал настоящий пользователь, он узнает его секретный ключ и сможет потратить его биткойны. Вероятность самостоятельной генерации злоумышленником входных данных для существующего обязательства пропорциональна стойкости хеш-алгоритма. Для безопасного 160-битного алгоритма, например HASH160, вероятность составляет 1 к 2^{160}. Это и есть *атака по прообразу* (*preimage attack*).

Взломщик также может попытаться сгенерировать два разных входа (например, скрипты погашения), которые создают одно и то же обязательство. Для адресов, полностью созданных одной стороной, вероятность генерации злоумышленником другого входа для существующего обязательства также составляет примерно 1 к 2^{160} для алгоритма HASH160. Это – *вторая атака по прообразу* (*second preimage attack*).

Однако ситуация меняется, когда злоумышленник может повлиять на первоначальное значение входа. Например, взломщик участвует в создании скрипта с несколькими подписями, где не нужно предоставлять свой открытый ключ до тех пор, пока он не узнает открытые ключи всех других участников. В этом случае надежность хеш-алгоритма снижается до его квадратного корня. Для HASH160 вероятность становится 1 к 2^{80}. Это – *коллизионная атака* (*collision attack*).

Чтобы оценить эту информацию в контексте, можно отметить, что по состоянию на начало 2023 года все майнеры Биткойна вместе взятые выполняли примерно 280 хеш-функций каждый час. Они используют хеш-функцию, отличную от HASH160, поэтому их оборудование не может создавать для нее коллизионные атаки. Однако существование сети Биткойн доказывает, что коллизионные атаки на 160-битные функции, такие как HASH160, практически осуществимы. Майнеры Биткойна потратили миллиарды долларов США на специальное оборудование, поэтому организация коллизионной атаки обойдется недешево, но есть организации, которые рассчитывают получить миллиарды долларов в биткойнах на адреса, сгенерированные в результате процессов с участием нескольких сторон, что может обеспечить прибыльность атаки.

Для предотвращения коллизионных атак есть хорошо отлаженные криптографические протоколы, но простое решение, не требующее от разработчиков кошельков специальных знаний, заключается в использовании более сильной хеш-функции. Эта возможность появилась благодаря последующим обновлениям Биткойна, и теперь новые адреса Биткойна обеспечивают как минимум 128 бит защиты от коллизий. Для выполнения 2^{128} хеш-операций всем нынешним майнерам Биткойна потребовалось бы около 32 миллиардов лет.

Хотя на сегодняшний день нет прямой угрозы для создания новых адресов P2SH, рекомендуем всем новым кошелькам использовать новые типы адресов для исключения угрозы коллизионных атак.

Адреса Bech32

В 2017 году протокол Биткойна был модернизирован. В результате обновления идентификаторы транзакций (txids) не могут быть изменены без согласия пользователя, расходующего средства (или кворума подписывающих лиц, если требуется несколько подписей). Модернизация, получившая название *Segregated Witness*, или сокращенно *Segwit* (буквально «изолированный свидетель»), также предоставила дополнительный объем для хранения данных транзакций в блоках и ряд других преимуществ. Однако для получения прямого доступа к преимуществам segwit пользователям пришлось принимать платежи на новые скрипты выхода.

Как указано в разделе «Адреса Pay to Script Hash», одним из преимуществ выхода по типу P2SH было отсутствие необходимости у отправителя (например, Алисы) знать детали скрипта, который использует получатель (например, Боб). Обновление segwit было разработано с учетом этого механизма, что дало пользователям возможность немедленно воспользоваться многими новыми преимуществами при использовании адреса P2SH. Но для получения Бобом

доступа ко всем преимуществам ему нужно, чтобы из кошелька Алисы ему заплатили с применением другого типа скрипта. Для поддержки новых скриптов кошелек Алисы необходимо обновить.

Сначала разработчики Биткойна предложили BIP142, который продолжал бы использовать base58check с новой версией байта, похожей на обновление P2SH. Но для обновления всех кошельков под новые скрипты с новой версией base58check потребовалось бы почти столько же работы, сколько для обновления под совершенно новый формат адресов. По этой причине несколько разработчиков Биткойна взялись за создание наилучшего формата адресов. Они определили несколько проблем с base58check:

- его смешанный формат неудобен для чтения вслух или расшифровки. Попробуйте прочесть один из содержащихся в этой главе устаревших адресов своему другу, чтобы он его расшифровал. Заметьте, как вам приходится добавлять к каждой букве слово «прописная» и «строчная». Также обратите внимание, что прописные и строчные варианты некоторых букв в почерке многих людей могут выглядеть совершенно одинаково;
- он способен обнаружить ошибки, но не помогает пользователям их исправить. Например, если вы случайно переставили два символа при ручном вводе адреса, ваш кошелек почти наверняка предупредит о наличии ошибки, но не поможет найти ее местоположение. В итоге вам придется потратить несколько неприятных минут на выявление ошибки;
- смешанный алфавит также требует дополнительного пространства для кодирования в QR-кодах, которые обычно используются для обмена адресами и счетами между кошельками. Это означает, что QR-коды должны быть крупнее при том же разрешении, иначе их будет сложнее быстро отсканировать;
- для поддержки новых функций протокола, таких как P2SH и segwit, требуется обновление каждого расходного кошелька. Хотя сами обновления не требуют большого количества кода, опыт показывает, что многие создатели кошельков заняты другими делами и порой откладывают обновление на годы. Это неблагоприятно сказывается на всех желающих использовать новые функции.

Создатели формата адресов для segwit нашли решение каждой из этих проблем в новом формате под названием bech32 (произносится с мягким «ch», как «besh thirty-two»). Сокращение «bech» образовано от аббревиатуры BCH. Она составлена из инициалов трех человек, в 1959–1960 годах открывших циклический код, который и лег в основу bech32. Число «32» означает количество символов в алфавите bech32 (по аналогии с числом 58 в base58check).

- В bech32 используются только цифры и один регистр букв (предпочтительно строчных). Несмотря на почти вдвое меньший размер алфавита по сравнению с используемым в base58check, адрес bech32 для скрипта хеша с открытым ключом (P2WPKH) лишь немного длиннее, чем старый адрес для эквивалентного скрипта P2PKH.
- bech32 может как обнаруживать, так и исправлять ошибки. В адресе ожидаемой длины он имеет математическую гарантию обнаружения любой

ошибки, влияющей на четыре символа или меньше; это надежнее, чем base58check. В более длинных ошибках он не сможет обнаружить их с вероятностью менее одного раза на миллиард, что примерно соответствует надежности base58check. Что особенно полезно, при наборе адреса с небольшим количеством ошибок программа может подсказать пользователю, где именно они были допущены, что позволит ему быстро исправить мелкие огрехи расшифровки. Образец адреса, введенного с ошибками, приведен в примере 4.3.

Пример 4.3 ❖ Определение опечаток в Bech32

```
Адрес:
bc1p9nh05ha8wrljf7ru236awn4t2x0d5ctkkywmv9sclnm4t0av2vgs4k3au7
```

Обнаруженные ошибки выделены жирным шрифтом и подчеркнуты. Сгенерировано с помощью демонстрационного декодера адресов bech32 (см. по ссылке: https://bitcoin.sipa.be/bech32/demo/demo.html).

- В bech32 предпочтительно вводить только строчные символы, но перед кодированием адреса в QR-код их можно заменить на прописные. Это дает возможность использовать специальный режим кодирования QR-кода, который занимает меньше места. Обратите внимание на разницу в размере и сложности двух QR-кодов для одного и того же адреса на рис. 4.9.

Рис. 4.9 ❖ Адрес bech32, закодированный в QR-коде в нижнем и верхнем регистрах

- bech32 пользуется преимуществами механизма обновления, созданного в рамках протокола segwit, чтобы кошельки пользователей могли оплачивать еще не используемые типы выхода. Идея в том, чтобы дать разработчикам возможность создавать уже сегодня кошельки с возможностью тратить средства на адрес bech32 и при этом обеспечить возможность тратить средства на адреса bech32 для тех, кто будет пользоваться новыми функциями после будущих обновлений протокола. Предполагалось, что нам больше никогда не придется проходить через циклы обновления всей системы, необходимые для полноценного использования P2SH и segwit.

Проблемы с адресами Bech32

Адреса bech32 оказались удачными во всем, кроме одной проблемы. Математические гарантии относительно их возможности обнаруживать ошибки действуют, только если длина адреса, который вводится в кошелек, совпадает с длиной исходного адреса. Если в процессе транскрипции добавить или удалить какие-либо символы, гарантия не сработает, и кошелек может отправить средства на неправильный адрес. Однако даже без этой гарантии вероятность того, что при добавлении или удалении символов пользователь получит строку с правильной контрольной суммой, была крайне мала, что обеспечивало безопасность средств пользователей.

К сожалению, выбор одной из констант в алгоритме bech32 привел к возможности легко добавлять или удалять букву «q» в предпоследней позиции адреса с окончанием на букву «p». В этих случаях можно также добавить или удалить букву «q» несколько раз. В некоторых случаях эта ошибка будет обнаружена контрольной суммой, однако будет пропущена намного чаще, чем ожидаемые один на миллиард для ошибок подстановки в bech32. Один из таких случаев приведен в примере 4.4.

Пример 4.4 ❖ Увеличение длины адреса bech32 без нарушения его контрольной суммы

```
Intended bech32 address:
bc1pqqqsq9txsqp

Incorrect addresses with a valid checksum:
bc1pqqqsq9txsqqqqp
bc1pqqqsq9txsqqqqqqp
bc1pqqqsq9txsqqqqqqqqp
bc1pqqqsq9txsqqqqqqqqqp
bc1pqqqsq9txsqqqqqqqqqqp
```

Для начальной версии segwit (версия 0) это не имело практического значения. Для выходов segwit версии v0 были определены только две допустимые длины: 22 и 34 байта. Они соответствуют адресам bech32 длиной 42 или 62 символа, так что кому-то пришлось бы 20 раз добавлять или убирать букву «q» из предпоследней позиции адреса bech32, чтобы отправить деньги на недействительный адрес без того, чтобы кошелек смог это обнаружить. Тем не менее это может стать проблемой для пользователей в будущем, если когда-либо будет реализовано обновление на основе segwit.

Bech32m

Хотя bech32 был хорош в версии segwit v0, разработчики не хотели излишне ограничивать размеры выходов в будущих его версиях. Без ограничений добавление или удаление одного «q» в адресе bech32 могло бы привести к случайной отправке пользователем своих денег на выход, который либо недоступен для расходования, либо может быть использован кем угодно (что давало бы возможность забрать эти биткойны кому угодно). Разработчики провели исчерпывающий анализ проблемы bech32 и выяснили, что для устранения проблемы достаточно изменить одну константу в алгоритме, в результате чего лю-

бая вставка или удаление до пяти символов будет оставаться незамеченной реже, чем один раз на миллиард.

Версия bech32 с единственной измененной константой известна как bech32 modified (bech32m). Все символы в адресах bech32 и bech32m для одних и тех же базовых данных идентичны, за исключением последних шести (контрольная сумма). Это означает, что для проверки контрольной суммы кошельку необходимо знать используемую версию. Однако оба типа адресов содержат внутренний байт проверки, который облегчает определение версии.

Ниже будут рассмотрены правила кодирования и разбора биткойн-адресов bech32 и bech32m, поскольку они включают в себя возможность анализа адресов bech32 и являются рекомендуемым в настоящее время форматом адресов для биткойн-кошельков.

Адреса bech32m начинаются с читаемой человеком части (Human Readable Part, HRP). В стандарте BIP173 есть правила для создания собственных HRP, но для Биткойна вам нужно знать только об уже выбранных HRP, приведенных в табл. 4.2.

Таблица 4.2. HRP в формате bech32 для Биткойна

Типы HRP	Сеть
bc	Основная сеть Биткойн
tb	Тестовая сеть Биткойн

За HRP идет разделитель в виде цифры «1». В более ранних предложениях для разделителя протоколов использовалось двоеточие, однако некоторые операционные системы и приложения, в которых можно дважды щелкнуть по слову для его выделения при копировании и вставке, не смогут продлить выделение до и после двоеточия. Число обеспечило работу выделения двойным щелчком мыши в любой программе, поддерживающей строки bech32m в целом (к которым относятся и другие числа). Цифра «1» была выбрана из-за того, что в других строках bech32 она не используется. Так удалось предотвратить случайную транслитерацию между цифрой «1» и строчной буквой «l».

Другая часть адреса в bech32m называется «часть данных» (data part). Эта часть состоит из трех элементов:

Witness version
: Один байт, который кодируется как один символ в биткойн-адресе bech32m сразу после разделителя. Эта буква обозначает версию segwit. Буква «q» соответствует кодировке «0» для segwit v0 – начальной версии, в которой были введены адреса bech32. Буква «p» означает кодировку «1» для версии segwit v1 (также называемой taproot), в которой начали использовать bech32m. Существует семнадцать возможных версий segwit, и для Биткойна обязательно, чтобы первый байт части данных bech32m декодировался в число от 0 до 16 (включительно).

Witness program
: От 2 до 40 байт. Для версии segwit v0 эта witness program должна иметь размер 20 или 32 байта; другая длина не допускается. Для segwit v1 единственная определенная длина на момент написания этой книги составляла 32 байта, но другие значения длины могут быть определены позже.

Checksum (*Контрольная сумма*)

Ровно 6 символов. Она создается с помощью кода BCH – тип кода для исправления ошибок (хотя для адресов Биткойна, как будет показано далее, контрольная сумма необходима только для обнаружения ошибок, но не для их исправления).

Эти правила можно наглядно проиллюстрировать на примере создания адресов bech32 и bech32m. Во всех следующих примерах используется эталонный код bech32m для Python (https://github.com/sipa/bech32/tree/master/ref).

Прежде всего нужно создать четыре скрипта выхода, по одному для каждого из различных выходов segwit, используемых на момент публикации книги, и один для будущей версии segwit, которая еще не имеет своего обозначения. Скрипты приведены в табл. 4.3.

Таблица 4.3. Скрипты для различных типов выходов segwit

Тип выхода	Пример скрипта
P2WPKH	OP_0 2b626ed108ad00a944bb2922a309844611d25468
P2WSH	OP_0 648a32e50b6fb7c5233b228f60a6a2ca4158400268844c4bc295ed5e8c3d626f
P2TR	OP_1 2ceefa5fa770ff24f87c5475d76eab519eda6176b11dbe1618fcf755bfac5311
Будущий вариант	OP_16 0000

Для выхода P2WPKH в witness program содержится обязательство, построенное точно так же, как и обязательство для выхода P2PKH, как описано в разделе «Устаревшие адреса для P2PKH». Открытый ключ передается в хеш-функцию SHA256. Полученный 32-байтовый дайджест передается в хеш-функцию RIPEMD-160. Дайджест этой функции (обязательство) помещается в witness program.

Для вывода хеша скрипта pay to witness (P2WSH) алгоритм P2SH не используется. Вместо этого нужно обработать скрипт, передать его в хеш-функцию SHA256 и использовать 32-байтовый дайджест этой функции в witness program. Для P2SH полученный из SHA256 дайджест повторно хеширован с помощью RIPEMD-160, но в некоторых случаях это может быть небезопасно; подробности см. в разделе «Коллизионные атаки на P2SH». В результате применения SHA256 без RIPEMD-160 обязательства P2WSH имеют размер 32 байта (256 бит) вместо 20 байт (160 бит).

Для выхода pay-to-taproot (P2TR) точка witness program находится на кривой secp256k1. Это может быть простой открытый ключ, но в большинстве случаев это должен быть открытый ключ с обязательством предоставить некоторые дополнительные данные. Подробнее об обязательствах будет рассказано в разделе «Taproot».

Для примера будущей версии segwit используется максимально возможный номер версии segwit (16) и минимально допустимая witness program (2 байта) с нулевым значением.

Теперь, когда известны witness version и witness program, можно преобразовать каждый из них в адрес bech32. Чтобы быстро сгенерировать эти адреса и рассмотреть происходящее более подробно, можно воспользоваться справочной библиотекой bech32m для Python:

```
$ github="https://raw.githubusercontent.com"
$ wget $github/sipa/bech32/master/ref/python/segwit_addr.py

$ python
>>> from segwit_addr import *
>>> from binascii import unhexlify

>>> help(encode)
encode(hrp, witver, witprog)
    Encode a segwit address.

>>> encode('bc', 0, unhexlify('2b626ed108ad00a944bb2922a309844611d25468'))
'bc1q9d3xa5gg45q2j39m9y32xzvygcgay4rgc6aaee'
>>> encode('bc', 0,
unhexlify('648a32e50b6fb7c5233b228f60a6a2ca4158400268844c4bc295ed5e8c3d626f'))
'bc1qvj9r9egtd7mu2gemy28kpf4zefq4ssqzdzzycj7zjhk4arpavfhsct5a3p'
>>> encode('bc', 1,
unhexlify('2ceefa5fa770ff24f87c5475d76eab519eda6176b11dbe1618fcf755bfac5311'))
'bc1p9nh05ha8wrljf7ru236awm4t2x0d5ctkkywmu9sclnm4t0av2vgs4k3au7'
>>> encode('bc', 16, unhexlify('0000'))
'bc1sqqqqkfw08p'
```

Если открыть файл *segwit_addr.py* и посмотреть, что делает код, то первое, что можно заметить, – что единственным различием между bech32 (используемым для segwit v0) и bech32m (используемым для более поздних версий segwit) является константа:

```
BECH32_CONSTANT = 1
BECH32M_CONSTANT = 0x2bc830a3
```

Далее следует код, создающий контрольную сумму. На последнем этапе создания контрольной суммы соответствующая константа добавляется к значению с помощью операции `xor`. Это значение – единственное различие между bech32 и bech32m.

После создания контрольной суммы каждый 5-битный символ в части данных (включая witness version, witness program и контрольную сумму) преобразуется в буквенно-цифровые символы.

Для обратного декодирования в скрипт выхода все делается в обратном порядке. Сначала с помощью библиотеки ссылок нужно декодировать два адреса:

```
>>> help(decode)
decode(hrp, addr)
    Decode a segwit address.

>>> _ = decode("bc", "bc1q9d3xa5gg45q2j39m9y32xzvygcgay4rgc6aaee")
>>> _[0], bytes(_[1]).hex()
(0, '2b626ed108ad00a944bb2922a309844611d25468')
>>> _ = decode("bc",
        "bc1p9nh05ha8wrljf7ru236awm4t2x0d5ctkkywmu9sclnm4t0av2vgs4k3au7")
>>> _[0], bytes(_[1]).hex()
(1, '2ceefa5fa770ff24f87c5475d76eab519eda6176b11dbe1618fcf755bfac5311')
```

В результате получается witness version и witness program. Их можно вставить в шаблон скрипта выхода:

```
<version> <program>
```

Например:

```
OP_0 2b626ed108ad00a944bb2922a309844611d25468
OP_1 2ceefa5fa770ff24f87c5475d76eab519eda6176b11dbe1618fcf755bfac5311
```

Одна из возможных ошибок, о которой следует знать, заключается в использовании witness version с `0` для OP_0, хотя она имеет байт 0x00, притом что witness program с `1` использует OP_1, которая имеет байт 0x51. В witness version со 2 по 16 используются байты с 0x52 по 0x60 соответственно.

При выполнении кодирования или декодирования bech32m настоятельно рекомендуется использовать тестовые векторы, представленные в BIP350. Кроме того, необходимо убедиться, что ваш код соответствует тестовым векторам, относящимся к будущим версиям segwit, которые еще не определены. Это поможет сделать ваши программы пригодными для работы в течение многих лет, даже при отсутствии невозможности добавить поддержку новых функций Биткойна по мере их появления.

Форматы секретных ключей

Секретный ключ может быть представлен в нескольких различных форматах, каждый из которых соответствует одному и тому же 256-битному числу. В табл. 4.4 приведено несколько распространенных форматов, используемых для представления секретных ключей. Различные форматы используются при различных обстоятельствах. Шестнадцатеричный и необработанный бинарный используются внутри программ и редко показываются пользователям. Формат WIF применяется для импорта/экспорта ключей между кошельками и часто для представления секретных ключей в виде QR-кода (или штрихкода).

Таблица 4.4. Формы представления секретных ключей (форматы кодирования)

Тип	Префикс	Описание
16-ричный (hex)	Нет	64 шестнадцатеричные цифры
WIF	5	Кодировка Base58check: base58 с префиксом версии 128 и 32-битной контрольной суммой
Сжатый WIF	K или L	Как указано выше, с добавлением суффикса 0x01 до кодирования

В табл. 4.5 показан секретный ключ в нескольких различных форматах.

Таблица 4.5. Пример: один и тот же ключ, разные форматы

Формат	Секретный ключ
16-ричный	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Сжатый WIF	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Значение форматов секретных ключей в настоящее время

В ранних версиях программных биткойн-кошельков при инициализации нового пользовательского кошелька создавался один или несколько независимых секретных ключей. После использования первоначального набора ключей кошелек мог генерировать дополнительные секретные ключи. Каждый секретный ключ можно было экспортировать или импортировать. Каждый раз, когда генерировались или импортировались новые секретные ключи, приходилось создавать новую резервную копию кошелька.

Позже в биткойн-кошельках появились детерминированные кошельки, где все секретные ключи генерируются из одного начального значения. Для обычного использования в блокчейне такие кошельки нужно резервировать только один раз. Однако если пользователь экспортирует один секретный ключ из такого кошелька, а злоумышленник получит этот ключ и некоторые несекретные данные о данном кошельке, он потенциально сможет вычислить любой секретный ключ в этом кошельке, что даст возможность украсть все средства из него. Кроме того, ключи нельзя импортировать в детерминированные кошельки. Это означает, что почти ни один современный кошелек не поддерживает возможность экспорта или импорта отдельных ключей. Информация в этом разделе представляет интерес в основном для тех, кому нужна совместимость с ранними версиями биткойн-кошельков.

Дополнительные сведения см. в разделе «Иерархическая детерминированная (HD) генерация ключей (BIP32)».

Все это – разные способы отображения одного и того же числа, одного и того же секретного ключа. Они выглядят по-разному, но каждый из них может быть легко преобразован в любой другой формат.

Сжатые секретные ключи

Популярный термин «сжатый секретный ключ» (compressed private key) не совсем соответствует сути вопроса, так как при экспорте секретного ключа с WIF-сжатием он на самом деле получается даже на один байт *длиннее*, чем «несжатый» секретный ключ. Это происходит из-за добавления к секретному ключу однобайтового суффикса (в табл. 4.6 он показан как 01 в шестнадцатеричном формате). Это означает, что секретный ключ получен из более новой версии кошелька и должен использоваться исключительно для создания сжатых открытых ключей. Секретные ключи сами по себе не являются сжатыми и не могут быть сжаты. Термин «сжатый секретный ключ» означает «секретный ключ, из которого можно получить только сжатые открытые ключи». В то время как «несжатый секретный ключ» означает «секретный ключ, из которого можно получить только несжатые открытые ключи». Во избежание дальнейшей путаницы следует называть формат экспорта только «сжатый WIF» или «WIF» и избегать обозначения самого секретного ключа как «сжатого». В табл. 4.6 показан один и тот же ключ, закодированный в форматах WIF и WIF-сжатый.

В формате сжатого секретного ключа в шестнадцатеричном формате есть один дополнительный байт в конце (01 в 16-ричном формате). Хотя префикс

версии кодировки base58 одинаков (0x80) для WIF и WIF-сжатых форматов, добавление одного байта в конце числа приводит к тому, что первый символ кодировки base58 меняется с 5 на K или L. Можно считать это эквивалентом разницы в десятичной кодировке между числом 100 и числом 99. Хотя число 100 на одну цифру длиннее числа 99, оно также имеет префикс 1, а не префикс 9. При изменении длины меняется и префикс. В base58 префикс 5 меняется на K или L при увеличении длины числа на один байт.

Таблица 4.6. Пример: один и тот же ключ, разные форматы

Формат	Секретный ключ
16-ричный (hex)	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Сжатый 16-ричный (hex-compressed)	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
Сжатый WIF (WIF-compressed)	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Важно помнить, что эти форматы не являются взаимозаменяемыми. В более новом кошельке, который использует сжатые открытые ключи, секретные ключи будут экспортироваться только в формате WIF со сжатием (с префиксом K или L). Если кошелек относится к более старой реализации и не использует сжатые открытые ключи, секретные ключи будут экспортироваться лишь в формате WIF (с префиксом 5). В данном случае цель состоит в передаче кошельку, импортирующему эти секретные ключи, сигнала о том, должен ли он искать в блокчейне сжатые или несжатые секретные ключи и адреса.

Если биткойн-кошелек умеет работать со сжатыми открытыми ключами, он будет использовать их во всех транзакциях. Секретные ключи в кошельке будут применяться для получения точек открытого ключа на кривой и будут сжаты. Сжатые открытые ключи будут использоваться для создания адресов Биткойна для транзакций. При экспорте секретных ключей из нового кошелька, в котором применяются сжатые открытые ключи, происходит модификация WIF с добавлением однобайтового суффикса 01 к секретному ключу. Полученный секретный ключ с кодировкой base58check называется «сжатым WIF» и начинается с буквы K или L, а не с «5», как в случае с WIF-кодированными (несжатыми) ключами от старых кошельков.

Расширенные ключи и адреса

В следующих разделах рассматриваются расширенные формы ключей и адресов, такие как престижные адреса и бумажные кошельки.

Престижные адреса

Престижные адреса (vanity addresses) представляют собой допустимые адреса Биткойна с сообщениями, которые может прочитать человек. Например,

1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 – это корректный адрес, буквы которого образуют слово «Love» в качестве первых четырех символов base58. Для создания престижных адресов потребуется сгенерировать и проверить миллиарды секретных ключей-кандидатов, пока не будет найден биткойн-адрес с нужным набором букв. Несмотря на некоторые оптимизации алгоритма генерации престижных адресов, процесс, по сути, включает в себя случайный выбор секретного ключа, получение открытого ключа, получение биткойн-адреса и проверку его соответствия желаемому шаблону престижности. Это повторяется миллиарды раз до получения совпадения.

После нахождения престижного адреса, соответствующего требуемому образцу, секретный ключ, на основе которого он был получен, может быть использован владельцем для применения биткойнов точно так же, как и любой другой адрес. Престижные адреса не менее и не более безопасны, чем любые другие. Они используют ту же криптографию эллиптических кривых (Elliptic Curve Cryptography, ECC) и безопасный хеш-алгоритм (Secure Hash Algorithm, SHA), что и любые другие адреса. Найти секретный ключ адреса, начинающегося с престижного адреса, не сложнее, чем любого другого адреса.

Итак, Евгения – глава детской благотворительной организации, работающей на Филиппинах. Допустим, Евгения организует сбор средств и хочет использовать престижный биткойн-адрес для рекламы сбора средств. Евгения создаст престижный адрес, начинающийся с «1Kids», чтобы прорекламировать сбор средств для детской благотворительности. Теперь разберемся, как будет создаваться этот престижный адрес и что это значит для безопасности благотворительной организации Евгении.

Генерация престижных адресов

Важно понимать, что биткойн-адрес – это просто число, представленное символами алфавита base58. Поиск шаблона вида «1Kids» можно рассматривать как поиск адреса в диапазоне от 1Kids11111111111111111111111111111111 до 1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz. В этом диапазоне примерно 58^{29} (около $1{,}4\times10^{51}$) адресов, и все они начинаются с «1Kids». В табл. 4.7 показан диапазон адресов с префиксом 1Kids.

Таблица 4.7. Диапазон престижных адресов, начинающихся с «1Kids»

От	1Kids11111111111111111111111111111
	1Kids11111111111111111111111111112
	1Kids11111111111111111111111111113
	…
До	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzz

В качестве примера можно рассмотреть шаблон «1Kids» в виде числа и посмотреть, как часто он встречается в биткойн-адресах (см. табл. 4.8). Обычный настольный компьютер без какого-либо специализированного оборудования может перебирать около 100 000 ключей в секунду.

Таблица 4.8. Частота появления престижного образца и среднее время его поиска на ПК

Длина	Образец	Частота	Среднее время поиска
1	1K	1 из 58 ключей	< 1 миллисекунды
2	1Ki	1 из 3364 ключей	50 миллисекунд
3	1Kid	1 из 195 000 ключей	< 2 секунд
4	1Kids	1 из 11 млн ключей	1 минута
5	1KidsC	1 из 656 млн ключей	1 час
6	1KidsCh	1 из 38 млрд ключей	2 дня
7	1KidsCha	1 из 2,2 трлн ключей	3–4 месяца
8	1KidsChar	1 из 128 трлн ключей	13–18 лет
9	1KidsChari	1 из 7 квадрильонов ключей	800 лет
10	1KidsCharit	1 из 400 квадрильонов ключей	46 000 лет
11	1KidsCharity	1 из 23 квинтильонов ключей	2,5 млн лет

Евгения, конечно, не станет в ближайшее время создавать престижный адрес «1KidsCharity», даже если у нее будет доступ к нескольким тысячам компьютеров. Каждый дополнительный символ увеличивает сложность в 58 раз. Шаблоны с более чем семью символами обычно находят с помощью специализированного оборудования, такого как заказные рабочие станции с несколькими графическими процессорами (GPU). Поиск престижных адресов на GPU-системах происходит на много порядков быстрее, чем на системах с процессорами общего назначения.

Другой способ найти престижный адрес – поручить эту работу пулу «майнеров престижа». Пул престижа (vanity pool, https://bitcointalk.org/index.php?topic=84569.0) представляет собой сервис, с помощью которого обладатели быстрого оборудования могут зарабатывать биткойны, отыскивая престижные адреса для других. За определенную плату Евгения может поручить поиск престижного адреса из семи символов и получить результаты за несколько часов, вместо того чтобы месяцами заниматься поиском на одном процессоре.

Генерация престижного адреса – это упражнение по перебору: ввод случайного ключа, проверка полученного адреса на соответствие требуемому шаблону, повтор до достижения успеха.

Безопасность и конфиденциальность престижного адреса

Престижные адреса пользовались популярностью в первые годы существования Биткойна, но по состоянию на 2023 год практически полностью вышли из употребления. У этой тенденции есть две вероятные причины:

Детерминированные кошельки

Как говорилось в разделе «Коды восстановления», для создания резервной копии каждого ключа в большинстве современных кошельков достаточно записать несколько слов или символов. Это делается путем вычисления каждого ключа в кошельке из этих слов или символов с помощью детерминированного алгоритма. Использование престижных адресов в детерминированном кошельке невозможно, если только пользователь не создает резервные

копии дополнительных данных для каждого созданного им престижного адреса. Более того, большинство кошельков, использующих детерминированную генерацию ключей, просто не позволяют импортировать секретный ключ или настройку ключа из «престижного генератора».

Предотвращение повторного использования адресов
Использование престижного адреса для получения нескольких платежей на один и тот же адрес устанавливает связь между всеми этими платежами. Такое решение может быть приемлемым для Евгении, если ее некоммерческая организация в любом случае должна отчитываться о своих доходах и расходах перед налоговыми органами. Однако подобный подход снижает конфиденциальность людей, которые либо платят Евгении, либо получают от нее платежи. Например, Алиса захочет сделать пожертвование анонимно, а Боб не захочет, чтобы другие его клиенты узнали о предоставлении скидки для Евгении.

Вряд ли в будущем появится много престижных адресов без решения вышеперечисленных проблем.

Бумажные кошельки

Бумажные кошельки (paper wallets) представляют собой секретные ключи, напечатанные на бумаге. Часто для удобства в бумажном кошельке также указывается соответствующий биткойн-адрес, но это необязательно, поскольку его можно вычислить из секретного ключа.

Бумажные кошельки – это СОВЕРШЕННО УСТАРЕВШАЯ технология, которая опасна для большинства пользователей. При их создании существует множество мелких ловушек, не последней из которых является возможность взлома кода генератора с помощью «черного хода» (back door). Многие биткойны были украдены именно так. Бумажные кошельки показаны здесь только в информационных целях и не подлежат использованию для хранения биткойнов. Используйте код восстановления для резервного копирования ключей, возможно, с аппаратным устройством подписи для хранения ключей и подписания транзакций. НЕ ИСПОЛЬЗУЙТЕ БУМАЖНЫЕ КОШЕЛЬКИ.

Бумажные кошельки бывают разных дизайнов и размеров и имеют множество различных функций. На рис. 4.10 показан пример бумажного кошелька.

Некоторые из них предполагают вручение в качестве подарка и имеют сезонную тематику, например рождественскую или новогоднюю. Другие предназначены для хранения в банковском хранилище или сейфе, при этом секретный ключ каким-либо образом скрыт – либо с помощью непрозрачных наклеек, либо сложен и запечатан клейкой фольгой, защищающей от несанкционированного доступа. Другие модели имеют дополнительные копии ключа и адреса в виде съемных корешков, похожих на корешки билетов, что позволяет хранить несколько копий для защиты от пожара, наводнения или других стихийных бедствий.

От первоначальной концепции системы Биткойн с открытым ключом до современных адресов и скриптов, таких как bech32m и pay to taproot, а также адресов для будущих обновлений Биткойна – теперь вы знаете, как протокол

Биткойн дает пользователям возможность идентифицировать кошельки для получения платежей. Но когда платеж поступает на ваш кошелек, вам захочется быть уверенным в сохранении доступа к этим деньгам, даже если с данными вашего кошелька что-то случится. В следующей главе на примере биткойн-кошельков будет рассказано о том, как они защищают свои средства от различных угроз.

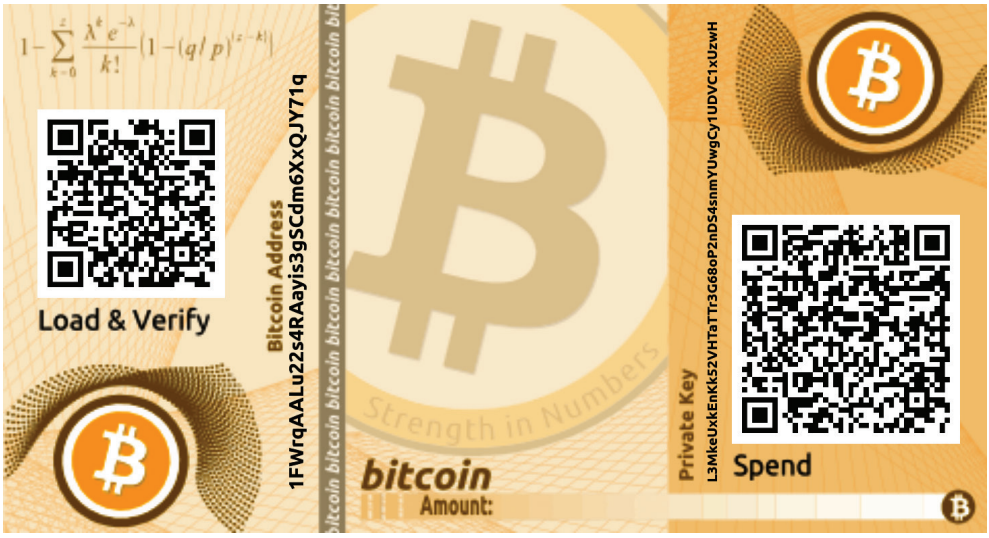


Рис. 4.10 ❖ Пример простого бумажного кошелька

Глава 5

Восстановление кошелька

Создание попарных секретных и открытых ключей – важнейший процесс, позволяющий биткойн-кошелькам получать и тратить биткойны. Но потеря доступа к секретному ключу лишает возможности тратить биткойны, полученные на соответствующий открытый ключ. Разработчики кошельков и протоколов на протяжении многих лет работали над созданием систем, обеспечивающих пользователям возможность восстановления доступа к своим биткойнам в случае возникновения проблем без ущерба для безопасности в остальное время.

В этой главе рассмотрены различные методы предотвращения потери данных, используемые в кошельках для защиты денег. Некоторые решения практически не имеют недостатков и повсеместно используются современными кошельками. Такие решения можно просто рекомендовать в качестве лучшего примера. Другие решения имеют как преимущества, так и недостатки, что вынуждает разработчиков разных кошельков идти на компромиссы. В этих случаях будут описаны имеющиеся варианты.

Независимая генерация ключей

Наличные деньги хранятся в «физических» кошельках, поэтому нет ничего удивительного в том, что многие ошибочно полагают, будто в биткойн-кошельках хранятся биткойны. На самом деле в том, что многие называют биткойн-кошельком – а мы называем *базой данных кошельков* (*wallet database*) для отличия от приложений для кошельков, содержатся только ключи. Эти ключи связаны с биткойнами, записанными в блокчейне. Доказав полным узлам Биткойна, что вы контролируете ключи, вы можете тратить связанные с ними биткойны.

В базах данных простых кошельков содержатся как открытые ключи, на которые поступают биткойны, так и секретные ключи, с помощью которых создаются подписи, необходимые для авторизации расходов этих биткойнов. Базы данных других кошельков могут содержать только открытые ключи или только отдельные секретные ключи, необходимые для авторизации транзакции. Приложения этих кошельков создают необходимые подписи при помощи внешних инструментов, таких как аппаратные устройства подписи или другие кошельки в схеме с несколькими подписями.

Приложение кошелька может самостоятельно генерировать каждый из ключей кошелька, которые оно впоследствии будет использовать, как показано на рис. 5.1, в виде коллекции независимо сгенерированных ключей, хранящихся в базе данных кошельков. Все первоначальные приложения биткойн-кошельков так и делали, но для этого пользователям приходилось создавать резервные копии базы данных кошельков каждый раз, когда они генерировали и распространяли новые ключи. Это могло происходить так часто, как и при генерации нового адреса для получения нового платежа. Если вовремя не создать резервную копию базы данных кошелька, пользователь мог потерять доступ к любым средствам, полученным на несозданные резервные копии ключей.

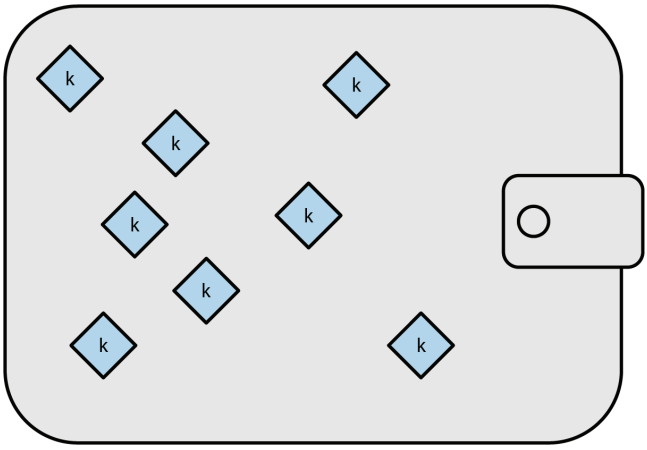


Рис. 5.1 ❖ Недетерминированная генерация ключей

Для каждого независимо сгенерированного ключа пользователю требовалось создать резервную копию размером около 32 байт плюс служебные данные. Некоторые пользователи и программы для работы с кошельками пытались минимизировать объем резервируемых данных путем использования только одного ключа. Несмотря на безопасность такого подхода, он значительно снижает конфиденциальность пользователя и всех тех, с кем он совершает сделки. Пользователи, которые дорожат конфиденциальностью, своей и своих коллег, создавали новые пары ключей для каждой транзакции. Таким образом, создавались базы данных кошельков, резервное копирование которых можно было обеспечить только с помощью цифровых носителей.

Современные приложения кошельков не генерируют ключи самостоятельно, а извлекают их из одного случайного «числа-первоисточника», или seed-числа (seed, в буквальном переводе «семя»), с помощью повторяющегося (детерминированного) алгоритма.

Детерминированная генерация ключей

При одинаковых входных данных хеш-функция всегда выдает один и тот же результат, но если входные данные хоть немного изменены, результат будет

другим. Если функция криптографически надежна, никто не сможет предсказать новый выход – даже если он знает новый вход.

Это позволяет использовать единственное случайное значение и преобразовывать его в практически неограниченное число якобы случайных значений. Более того, при последующем использовании одной и той же хеш-функции с одним и тем же входом (то самое вышеупомянутое seed-число) будут получены одни и те же якобы случайные значения:

```
# Collect some entropy (randomness)
$ dd if=/dev/random count=1 status=none | sha256sum
f1cc3bc03ef51cb43ee7844460fa5049e779e7425a6349c8e89dfbb0fd97bb73 -

# Set our seed to the random value
$ seed=f1cc3bc03ef51cb43ee7844460fa5049e779e7425a6349c8e89dfbb0fd97bb73

# Deterministically generate derived values
$ for i in {0..2} ; do echo "$seed + $i" | sha256sum ; done
50b18e0bd9508310b8f699bad425efdf67d668cb2462b909fdb6b9bd2437beb3 -
a965dbcd901a9e3d66af11759e64a58d0ed5c6863e901dfda43adcd5f8c744f3 -
19580c97eb9048599f069472744e51ab2213f687d4720b0efc5bb344d624c3aa -
```

Если использовать полученные значения в качестве секретных ключей, то позже можно сгенерировать точно такие же секретные ключи с помощью начального значения и ранее использованного алгоритма. Пользователю детерминированной генерации ключей можно создать резервную копию каждого ключа в своем кошельке, просто записав число-первоисточник (seed) и ссылку на используемый детерминированный алгоритм. Например, даже если у Алисы есть миллион биткойнов, полученных на миллион разных адресов, все, что ей нужно сделать для восстановления доступа к этим биткойнам, – это

```
f1cc 3bc0 3ef5 1cb4 3ee7 8444 60fa 5049
e779 e742 5a63 49c8 e89d fbb0 fd97 bb73
```

Логическая схема основной последовательной детерминированной генерации ключей показана на рис. 5.2. Однако в современных приложениях для работы с кошельками этот процесс реализован более продуманно: открытые ключи могут быть получены отдельно от соответствующих им секретных ключей, что позволяет хранить секретные ключи более надежно, чем открытые.

Деривация открытого дочернего ключа

В разделе «Открытые ключи» описан процесс создания открытого ключа из секретного ключа с помощью криптографии на эллиптической кривой (Elliptic Curve Cryptography, ECC). Хотя операции на эллиптической кривой не слишком понятны на интуитивном уровне, они похожи на операции сложения, вычитания и умножения, используемые в обычной арифметике. Другими словами, открытый ключ можно складывать или вычитать из него, а также выполнять его умножение. В разделе «Открытые ключи» описана операция генерации открытого ключа (K) из секретного ключа (k) с помощью точки генерации (G):

$$K = k \times G.$$

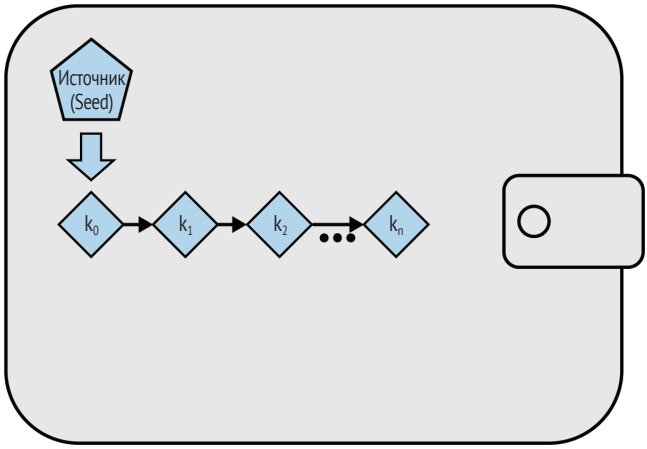


Рис. 5.2 ❖ Детерминированная генерация ключей

Можно создать производную пару ключей, называемую дочерней парой ключей (child key pair), просто добавив одно и то же значение к обеим сторонам уравнения:

$K + (123 \times G) == (k + 123) \times G.$

В уравнениях этой книги одинарный знак равенства используется для таких операций, как $K = k \times G$, где вычисляется значение переменной. Двойной знак равенства применяется для обозначения эквивалентности обеих сторон уравнения или для того, чтобы операция возвращала false (не истину – not true), если обе стороны не эквивалентны.

Из этого следует интересный вывод: добавление 123 к открытому ключу может быть выполнено с использованием полностью открытой информации. Например, Алиса генерирует открытый ключ *K* и передает его Бобу. Боб не знает секретного ключа, но ему известна глобальная константа *G*, поэтому он может добавить любое значение к открытому ключу для получения производного открытого дочернего ключа. Если затем он сообщит Алисе добавленное к открытому ключу значение, она сможет добавить то же значение к секретному ключу и получить производный секретный ключ, соответствующий созданному Бобом открытому дочернему ключу.

Иными словами, деривация (создание) дочерних открытых ключей возможна даже без информации о родительском секретном ключе. Значение, добавляемое к открытому ключу, называется «*твик ключа*» (*key tweak*). Если для генерации твиков ключа выбран детерминированный алгоритм, любой пользователь, не знающий секретного ключа, может создать практически неограниченную последовательность дочерних открытых ключей на основе одного родительского открытого ключа. Тот, кто контролирует секретный родительский ключ, может затем использовать те же самые твики для создания всех соответствующих секретных ключей.

Эта методика обычно используется для разделения внешних интерфейсов (фронтендов) приложений кошелька (которые не требуют секретных ключей)

и операций подписи (которые требуют секретных ключей). Например, внешний интерфейс Алисы распространяет ее открытые ключи среди желающих ей заплатить. Позже, когда захочется потратить полученные деньги, она может передать использованные ею твики ключей аппаратному устройству подписи (иногда его еще неправильно называют аппаратным кошельком – hardware wallet), которое надежно хранит ее исходный секретный ключ. Устройство аппаратной подписи использует твики для получения необходимых дочерних секретных ключей и с их помощью подписывает транзакции, возвращая их менее защищенному фронтенду для передачи в сеть Биткойна.

При извлечении открытого дочернего ключа может быть создана линейная последовательность ключей, подобная показанной на рис. 5.2. Однако современные приложения для создания кошельков используют еще один прием, позволяющий получить дерево ключей вместо одной последовательности, как описано в следующем разделе.

Иерархическая детерминированная (HD) генерация ключей (BIP32)

Каждый известный современный биткойн-кошелек по умолчанию использует иерархическую детерминированную (Hierarchical Deterministic, HD) генерацию ключей. Этот стандарт, описанный в протоколе BIP32, использует детерминированную генерацию ключей и опциональное получение открытых дочерних ключей с помощью алгоритма создания дерева ключей. В этом дереве любой ключ может быть родительским для последовательности дочерних ключей, а любой из этих дочерних ключей может быть родительским по отношению к другой последовательности дочерних ключей («внуков» исходного ключа). Не существует каких-либо условных ограничений на глубину дерева. Такая древовидная структура показана на рис. 5.3.

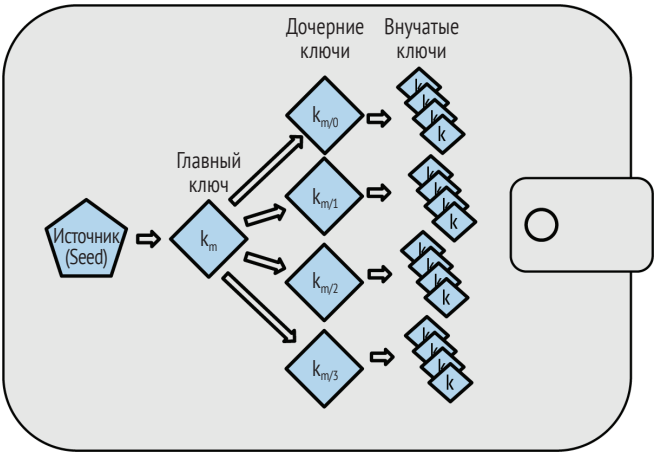


Рис. 5.3 ❖ HD-кошелек: дерево ключей, сгенерированных из одного seed-числа

Древовидную структуру можно использовать для отображения дополнительного структурного содержания, например когда определенная ветвь вложенных ключей (субключей – subkeys) используется для приема входящих платежей, а другая ветвь – для приема сдачи от исходящих платежей. Ветви ключей также можно использовать в корпоративных системах, распределяя функции по отделам, дочерним компаниям, определенным функциям или категориям учета.

Подробнее о HD-кошельках будет рассказано в разделе «Создание HD-кошелька из seed-числа».

Seed-числа и коды восстановления

HD-кошельки являются очень мощным механизмом для управления множеством ключей, полученных из одного числа-первоисточника. В случае повреждения или потери базы данных кошелька все секретные ключи для него можно сгенерировать заново с исходным seed-числом. Но если это «семя» попадет к кому-то другому, он также сможет сгенерировать все секретные ключи. Это позволит ему украсть все биткойны из кошелька с единственной подписью и снизит степень защиты биткойнов в кошельках с несколькими подписями. В этом разделе рассмотрены различные коды восстановления, которые позволяют сделать резервное копирование более простым и безопасным.

Хотя seed-числа – это большие случайные числа, обычно длиной от 128 до 256 бит, в большинстве кодов восстановления используются понятные человеку слова. В значительной степени смысл применения слов заключается в стремлении сделать код восстановления легко запоминающимся. Для примера можно взять код восстановления, закодированный с использованием шестнадцатеричной системы и слов в примере 5.1.

Пример 5.1 ❖ Число-первоисточник, закодированное в шестнадцатеричной системе и словами на английском языке

```
Hex-encoded:
0C1E 24E5 9177 79D2 97E1 4D45 F14E 1A1A

Word-encoded:
army van defense carry jealous true
garbage claim echo media make crunch
```

В некоторых случаях запомнить код восстановления бывает очень полезно, например когда вы не можете переместить физические вещи (например, код восстановления, записанный на бумаге) без их изъятия или проверки посторонними лицами, которые могут украсть ваши биткойны. Однако чаще всего полагаться только на память весьма опасно:

- если вы забудете код восстановления и потеряете доступ к исходной базе данных кошелька, ваши биткойны будут потеряны для вас навсегда;
- если вы умрете или получите тяжелую травму, а у ваших наследников не будет доступа к базе данных вашего кошелька, они не смогут унаследовать ваши биткойны;

- если кто-то подозревает, что вы запомнили код восстановления для доступа к биткойнам, он может попробовать заставить вас раскрыть этот код. На момент написания данной книги автор статьи о биткойнах Джеймсон Лопп (Jameson Lopp) задокументировал (https://github.com/jlopp/physical-bitcoin-attacks/blob/master/README.md) более 100 физических нападений на предполагаемых владельцев биткойнов и других цифровых активов, включая как минимум три смертельных исхода и множество эпизодов с пытками, захватом в заложники или угрозами в адрес семьи.

Даже при использовании кода восстановления, разработанного для простого запоминания, настоятельно рекомендуем записать его.

На сегодняшний день широко используется несколько различных типов кодов восстановления.

BIP39

Самый популярный в последнее десятилетие метод генерации кодов восстановления BIP39 включает в себя получение случайной последовательности байтов, добавление к ней контрольной суммы и кодирование данных в серию из 12–24 слов, которые могут быть локализованы в соответствии с родным языком пользователя. Эти слова плюс дополнительная парольная фраза (passphrase) проходят через *функцию растяжения ключа* (*key-stretching function*), а полученный результат используется в качестве seed-числа. Коды восстановления BIP39 имеют некоторые недостатки, которые стараются устранить в более поздних версиях.

Electrum v2

Этот словесный код восстановления, используемый в кошельке Electrum (версия 2.0 и выше), имеет ряд преимуществ перед BIP39. Он не зависит от универсального списка слов, который должен быть реализован в каждой версии любой совместимой программы, к тому же его коды восстановления содержат номер версии, что повышает надежность и эффективность. Как и BIP39, он поддерживает дополнительную парольную фразу, которую в Electrum называют *seed-расширением* (*seed extension*), и применяет такую же функцию растягивания ключа.

Aezeed

Используется в кошельке LND. Это еще один код восстановления на основе слов, который предлагает ряд улучшений по сравнению с BIP39. Он имеет два номера версии: один – внутренний, устраняющий некоторые проблемы с обновлением приложений кошелька (как номер версии Electrum v2); другой номер версии – внешний, который может увеличиваться для изменения базовых криптографических свойств кода восстановления. В код восстановления также включена *дата создания кошелька* (*wallet birthday*) – ссылка на день, когда пользователь создал базу данных кошелька. Это позволяет процессу восстановления найти все средства, связанные с кошельком, без сканирования всего блокчейна, что особенно полезно для легких клиентов с упором на конфиденциальность. Поддерживается смена парольной фразы или изменение других аспектов кода восстановления без необходимости

переноса средств на новое seed-число – пользователю достаточно создать резервную копию нового кода восстановления. Одним из недостатков по сравнению с Electrum v2 является, как и в случае с BIP39, необходимость поддержки одного и того же списка слов в программах резервного копирования и восстановления.

Muun

Используется в кошельке Muun, который по умолчанию предполагает подпись транзакций несколькими ключами. Это несловарный код, который должен сопровождаться дополнительной информацией. В настоящее время Muun предоставляет ее в формате PDF. Этот код восстановления не имеет отношения к seed-числу и используется для расшифровки секретных ключей из PDF-файла. Несмотря на громоздкость по сравнению с кодами восстановления BIP39, Electrum v2 и Aezeed, он обеспечивает поддержку новых технологий и стандартов, которые все чаще встречаются в новых кошельках, например протокола Lightning Network (LN), дескрипторов скриптов выхода и мини-скриптов.

SLIP39

Преемник BIP39 от тех же авторов, SLIP39 позволяет распространять единый материал seed-числа с помощью нескольких кодов восстановления, которые могут храниться в разных местах (или у разных людей). При создании кодов восстановления можно указать их количество, необходимое для восстановления посевного материала. Например, можно создать пять кодов восстановления, но для восстановления посевного материала потребуется только три из них. В SLIP39 предусмотрена поддержка опциональной парольной фразы, которая зависит от универсального списка слов и не обеспечивает непосредственного управления версиями.

Во время написания этой книги была предложена новая система распространения кодов восстановления, имеющая сходство со SLIP39. С помощью Codex32 можно создавать и проверять коды восстановления, используя только печатные инструкции, ножницы, острый нож, латунные застежки (brass fasteners) и ручку, а также уединение и несколько часов свободного времени. Кроме того, любители компьютеров могут мгновенно создавать коды восстановления с помощью приложения на цифровом устройстве. Можно составить до 31 кода восстановления, которые будут храниться в разных местах, и указать количество таких кодов, необходимых для восстановления seed-числа. Поскольку Codex32 является новым предложением, его детали могут существенно измениться до выхода этой книги, поэтому читатели, заинтересованные в использовании распределенных кодов восстановления, могут самостоятельно ознакомиться с его текущим статусом.

Парольные фразы для кодов восстановления

При использовании BIP39, Electrum v2, Aezeed и SLIP39 может применяться дополнительная парольная фраза (passphrase). Если хранить эту кодовую фразу только в своей памяти, она будет иметь те же преимущества и недостатки, что и запоминание кода восстановления. Однако существуют и другие особенности, связанные с использованием парольной фразы для кода восстановления.

Три варианта (BIP39, Electrum v2 и SLIP39) позволяют не использовать опциональную кодовую фразу в контрольной сумме, которая применяется для защиты от

ошибок при вводе данных. Любая парольная фраза (в том числе и ее отсутствие) приведет к созданию seed-числа для дерева ключей BIP32, но это будут разные деревья. Разные парольные фразы приведут к разным ключам. Это может быть как положительным, так и отрицательным фактором в зависимости от ваших предпочтений.

- Положительный момент: если кто-то получит ваш код восстановления (но не вашу парольную фразу), он увидит корректное дерево ключей BIP32. Если вы готовы к такому повороту событий и отправили несколько биткойнов на дерево без парольной фразы, эти деньги будут украдены. Хотя кража части ваших биткойнов, конечно, неприятна, она также может стать для вас предупреждением о компрометации вашего кода восстановления, что даст вам возможность провести расследование и принять меры по исправлению ситуации. Способность создавать несколько парольных фраз для одного и того же кода восстановления, которые все выглядят достоверными, является одной из разновидностей правдоподобного отрицания.
- Отрицательный момент: если злоумышленника заставили сообщить код восстановления (с парольной фразой или без), но он не получил ожидаемого количества биткойнов, он может продолжить попытки выманить у вас деньги, пока вы не сообщите ему другую парольную фразу с доступом к большему количеству биткойнов. Разработка с учетом правдоподобного отрицания означает, что нет способа доказать злоумышленнику, что вы раскрыли всю свою информацию, поэтому он может продолжить попытки давления на вас даже после передачи ему всех биткойнов.
- Дополнительным минусом можно считать сокращение числа обнаруженных ошибок. Если при восстановлении из резервной копии ввести слегка неправильную парольную фразу, кошелек не сможет предупредить об ошибке. Если вы ждали поступления средств на баланс, вы сразу поймете, что дело неладно, когда приложение кошелька покажет нулевой баланс для восстановленного дерева ключей. Однако начинающие пользователи могут подумать, что их деньги безвозвратно потеряны, и совершить какую-нибудь глупость, например опустить руки и выбросить код восстановления. Или если на самом деле ожидался нулевой баланс, можно использовать приложение кошелька в течение многих лет после ошибки, пока в следующий раз при восстановлении с правильной ключевой фразой не появится нулевой баланс. Если не удастся определить, какую опечатку вы допустили ранее, ваши средства пропадут.

Система шифрования seed-чисел Aezeed, в отличие от остальных, проверяет подлинность опциональной парольной фразы и возвращает ошибку при указании неверного значения. Это исключает вариант правдоподобного отрицания, дает возможность обнаружить ошибку и подтвердить факт раскрытия парольной фразы.

Многие пользователи и разработчики расходятся во мнении о том, какой подход лучше: одни выступают за правдоподобное отрицание, другие предпочитают повышенную безопасность за счет обнаружения ошибок, которую получают начинающие пользователи и пользователи, находящиеся под принуждением. Есть основания полагать, что эти споры будут продолжаться в течение всего времени существования кодов восстановления.

Сохранение данных, не связанных с ключами

Самые важные данные в базе данных кошелька – это секретные ключи. При потере доступа к секретным ключам теряется возможность тратить свои биткойны. Детерминированные коды извлечения и восстановления ключей представляют собой достаточно надежное решение для резервного копирования и восстановления ваших ключей и управляемых ими биткойнов. Однако важно учитывать, что многие базы данных кошельков хранят не только ключи, но также предоставляемую пользователем информацию о каждой отправленной или полученной транзакции.

Например, когда Боб создает новый адрес для отправки счета Алисе, он добавляет метку к создаваемому адресу, чтобы отличить этот платеж от других, полученных им. Когда Алиса производит оплату по адресу Боба, она по той же причине отмечает транзакцию как оплату Бобу. Некоторые кошельки также добавляют к транзакциям другую полезную информацию, например текущий обменный курс, который может быть полезен для расчета налогов в некоторых юрисдикциях. Эти метки хранятся исключительно в самих кошельках, а не передаются в сеть, защищая конфиденциальность и не допуская попадания ненужных личных данных в блокчейн. В качестве примера см. табл. 5.1.

Таблица 5.1. История транзакций Алисы с пометкой каждой из них

Дата	Пометка	BTC
2023-01-01	Купила биткойны у Джо	+0,00100
2023-01-02	Заплатила Бобу за подкаст	–0,00075

Но поскольку метки адресов и транзакций хранятся только в базе данных кошелька каждого пользователя и не являются детерминированными, их невозможно восстановить с помощью только лишь кода восстановления. Если единственным способом восстановления является восстановление по seed-числам, то все, что увидит пользователь, – это список приблизительного времени транзакций и суммы биткойнов. В результате становится довольно сложно выяснить информацию о прошлых транзакциях. Представьте себе выписку по банковской или кредитной карте годичной давности, в которой указаны дата и сумма каждой транзакции, но отсутствует поле «описание».

Кошельки должны обеспечивать своим пользователям удобный способ резервного копирования данных об этих метках. Это кажется очевидным, но существует ряд распространенных приложений для кошельков, которые упрощают создание и использование кодов восстановления, но не предоставляют возможности резервного копирования или восстановления пользовательских записей.

Кроме того, приложениям-кошелькам может пригодиться стандартный формат для экспорта меток, чтобы их можно было использовать в других приложениях (например, в бухгалтерских программах). Стандарт для такого формата предложен в протоколе BIP329.

В приложениях кошелька, поддерживающих дополнительные протоколы, помимо основных функций Биткойна, может возникнуть необходимость или желание хранить другие данные. Например, по состоянию на 2023 год все

больше приложений добавляют поддержку отправки и получения транзакций через Lightning Network (LN). Хотя протокол LN предусматривает метод восстановления средств в случае потери данных под названием *«статическое резервирование канала»* (*static channel backups*), он не гарантирует результат. Если узел, к которому подключается ваш кошелек, обнаружит потерю данных, он сможет украсть ваши биткойны. Если он потеряет базу данных своего кошелька одновременно с потерей вашей базы данных и ни у кого из вас не будет необходимой резервной копии, вы оба потеряете свои средства.

Это означает, что пользователям и приложениям кошельков придется делать нечто большее, чем просто резервное копирование кода восстановления.

Одним из решений, которое используется рядом приложений для создания кошельков, является частое и автоматическое создание полных резервных копий базы данных кошельков, которые шифруются одним из ключей, полученных из seed-чисел. Ключи биткойна не должны быть угадываемыми, к тому же современные алгоритмы шифрования считаются очень надежными. Поэтому никто не сможет открыть зашифрованную резервную копию, за исключением тех, кто может генерировать seed-число. Это позволяет хранить резервную копию на компьютерах без соответствующего доверия, например на облачных хостингах или даже у случайных сетевых пиров.

В дальнейшем, в случае утери исходной базы данных кошелька, пользователь сможет ввести код восстановления в приложение кошелька для повторного получения своих seed-чисел. Затем приложение может извлечь последний файл резервной копии, заново сгенерировать ключ шифрования, расшифровать резервную копию, а также восстановить все метки пользователя и дополнительные данные протокола.

Резервное копирование путей извлечения ключей

В дереве ключей BIP32 порядка 4 млрд ключей первого уровня. У каждого из них может быть 4 млрд дочерних ключей, у каждого из которых может быть 4 млрд собственных дочерних ключей, и так далее. Приложение кошелька не сможет сгенерировать даже малую часть всех возможных ключей в дереве BIP32, а это значит, что для восстановления после потери данных необходимо знать не только код восстановления, алгоритм получения seed-числа (например, BIP39) и детерминированный алгоритм получения ключей (например, BIP32) – для этого также нужно знать, какие пути в дереве ключей использовало приложение вашего кошелька для генерации распространяемых им конкретных ключей.

Для решения этой проблемы было выбрано два варианта. Первый состоит в использовании стандартных путей. Каждый раз при изменениях в адресах, которые могут генерировать приложения кошельков, кто-то создает BIP для определения пути получения ключей. Например, BIP44 определяет m/44’/0’/0’ как путь, который следует использовать для ключей в скриптах P2PKH (унаследованный адрес). Приложение кошелька, применяющее этот стандарт, использует ключи по этому пути как при первом запуске, так и после восстановления с помощью кода восстановления. Такое решение называют *неявно заданными*,

или *косвенными*, *путями* (*implicit paths*). Несколько популярных неявно заданных путей, определенных в разных вариантах BIP, приведены в табл. 5.2.

Таблица 5.2. Неявные пути скриптов, определенные различными BIP

Стандарт	Скрипт	Путь BIP32
BIP44	P2PKH	m/44'/0'/0'
BIP49	Вложенный P2WPKH	m/49'/1'/0'
BIP84	P2WPKH	m/84'/0'/0'
BIP86	Единый (single-key) P2TR	m/86'/0'/0'

Второй вариант – дополнить информацию о пути кодом восстановления, чтобы четко указать, какой путь используется с теми или иными скриптами. Такое решение называют *явно заданными путями* (*explicit paths*).

Преимущество неявных путей – в отсутствии необходимости вести учет используемых вариантов. Если пользователь введет код восстановления в приложение-кошелек, которое он использовал ранее, той же версии или выше, оно автоматически сгенерирует ключи для тех же путей, которые использовались ранее.

Недостатком неявно заданных скриптов можно назвать отсутствие гибкости такого решения. При вводе кода восстановления приложение кошелька должно сгенерировать ключи для каждого поддерживаемого им пути и просканировать блокчейн на предмет транзакций с этими ключами – в противном случае оно может не выявить все транзакции пользователя. Это нерационально для кошельков с поддержкой множества функций, каждая из которых имеет свой собственный путь, но при этом используется лишь несколько из этих функций.

Для неявно заданных путей кодов восстановления, не содержащих номера версии, таких как BIP39 и SLIP39, новая версия приложения для кошелька без поддержки старого пути не сможет предупредить пользователя в процессе восстановления о том, что часть его средств может быть не найдена. Та же проблема возникает и в противоположном варианте, если пользователь будет вводить свой код восстановления в старое программное обеспечение: оно не найдет более поздние пути, по которым пользователь мог получить средства. Коды восстановления с указанием версии, такие как Electrum v2 и Aezeed, могут обнаружить факт ввода пользователем более старого или более нового кода восстановления и направить его на соответствующие ресурсы.

И последнее обстоятельство использования неявных путей заключается в том, что они могут содержать только универсальную информацию (например, стандартизированный путь) или полученную из seed-числа (например, ключи). Важную недетерминированную информацию, характерную для конкретного пользователя, невозможно восстановить с помощью кода восстановления. Например, Алиса, Боб и Кэрол получают средства, которые могут быть потрачены только при наличии подписей двух из трех пользователей. И хотя Алисе для расходования средств нужна только подпись Боба или Кэрол, для поиска их совместных средств в блокчейне ей понадобится оба их открытых ключа. Это означает, что каждый из них должен создать резервные копии открытых ключей для всех троих. По мере распространения мультиподписей

и других современных скриптов в Биткойне недостаточная гибкость неявных путей становится все более значимой.

Преимуществом явных путей является возможность точно указать ключи, которые следует использовать с теми или иными скриптами. Нет нужды поддерживать устаревшие скрипты, нет проблем с прямой или обратной совместимостью, а любая дополнительная информация (например, открытые ключи других пользователей) может быть представлена напрямую. Недостатком является необходимость резервного копирования дополнительной информации вместе с кодом восстановления. Дополнительная информация обычно не может поставить под угрозу безопасность пользователя, поэтому она не требует той же защиты, что и код восстановления, хотя она может снизить конфиденциальность и поэтому ее тоже необходимо защищать.

Почти все приложения кошельков, использующие явные пути, на момент написания этой книги применяли стандарт *дескрипторов скриптов выхода* (*output script descriptors*, для краткости их также называют просто «*дескрипторами*»), описанный в протоколах BIP 380, 381, 382, 383, 384, 385, 386 и 389. Дескрипторы описывают скрипт и ключи (или пути к ключам), которые будут с ним использоваться. Несколько примеров дескрипторов приведены в табл. 5.3.

Таблица 5.3. Примеры дескрипторов из документации Bitcoin Core (с сокращениями)

Дескриптор	Описание
`pkh(02c6…9ee5)`	Скрипт P2PKH для заданного открытого ключа
`sh(multi(2,022f…2a01,03ac…ccbe))`	Мультиподпись P2SH, требующая двух подписей соответственно двум указанным ключам
`pkh([d34db33f/44'/0'/0']xpub6ERA…RcEL/1/*)`	Скрипты P2PKH для BIP32 `d34db33f` с расширенным открытым ключом (xpub) на пути `M/44'/0'/0'`, то есть `xpub6ERA...RcEL`, с использованием ключей на `M/1/*` этого xpub

Приложения кошельков, разработанные только для скриптов с одной подписью, долгое время использовали неявные пути. Приложения кошельков для работы с несколькими подписями или другими сложными скриптами все чаще применяют поддержку явных путей с помощью дескрипторов. Приложения с поддержкой обоих вариантов обычно соответствуют стандартам для неявных путей, а также поддерживают дескрипторы.

Подробнее о технологическом стеке кошелька

Разработчики современных кошельков имеют возможность выбирать из множества различных технологий для создания и использования резервных копий, и каждый год появляются совершенно новые решения. Вместо подробного описания каждого из описанных ранее вариантов в этой главе основное внимание будет уделено распространенным технологиям, которые, по мнению авторов, будут наиболее широко использоваться в кошельках в начале 2023 года:

- коды восстановления BIP39;
- деривация (выведение) ключа BIP32 HD;
- неявные пути по типу BIP44.

Все эти стандарты применяются с 2014 года или раньше, и найти дополнительные ресурсы для их использования не составит труда. Впрочем, набравшись смелости, стоит изучить более современные стандарты, которые могут дать дополнительные возможности или повысить безопасность.

Коды восстановления BIP39

Коды восстановления BIP39 – это последовательности слов, которые обозначают (кодируют) случайное число, используемое в качестве начального (seed) числа для создания детерминированного кошелька. Этой последовательности слов достаточно для повторного создания seed-числа и, исходя из него, воссоздания всех производных ключей. При создании кошелька приложение с поддержкой детерминированных кошельков и кодом восстановления BIP39 выдаст пользователю последовательность из 12–24 слов. Эта последовательность является резервной копией кошелька и может быть использована для восстановления и повторного создания всех ключей в том же или любом другом совместимом приложении кошелька. Коды восстановления значительно упрощают процесс создания резервных копий, поскольку их легко прочитать и правильно расшифровать.

Коды восстановления часто путают с технологией brainwallet (буквально «кошелек для мозга»). Это не одно и то же. Основное отличие заключается в том, что brainwallet содержит слова, выбранные пользователем, в то время как коды восстановления создаются кошельком случайным образом и передаются пользователю. Это важное отличие делает коды восстановления гораздо более безопасными, поскольку люди – очень ненадежные источники случайных данных.

Следует отметить, что BIP39 – это одна из реализаций стандарта кода восстановления. Протокол BIP39 был предложен компанией, создавшей аппаратный кошелек Trezor, и он совместим со многими другими кошельками, хотя, конечно, не со всеми.

BIP39 определяет процесс создания кода восстановления и seed-числа, который описывается здесь, в девять шагов. Для наглядности процесс разделен на две части: шаги с 1 по 6 показаны в разделе «Генерация кода восстановления», а шаги с 7 по 9 – в разделе «От кода восстановления до seed-числа».

Генерация кода восстановления

Коды восстановления генерируются автоматически приложением кошелька с помощью стандартизированного процесса, определенного в BIP39. Кошелек начинает с источника энтропии, добавляет контрольную сумму, а затем сопоставляет энтропию со списком слов.

1. Создайте случайную последовательность (энтропию) длиной от 128 до 256 бит.
2. Создайте контрольную сумму случайной последовательности, взяв первые (длина энтропии/32) бита ее хеша SHA256.
3. Добавьте контрольную сумму в конец случайной последовательности.

4. Разделите результат на сегменты длиной 11 бит.
5. Свяжите каждое 11-битное значение со словом из заранее составленного словаря из 2048 слов.
6. Кодом восстановления является последовательность этих слов.

На рис. 5.4 представлено применение энтропии для генерации кода восстановления BIP39.

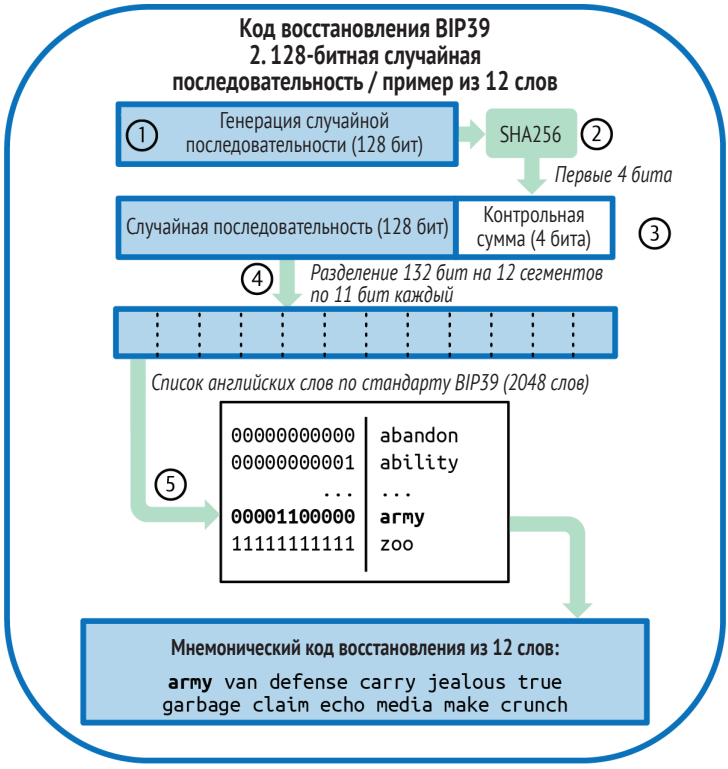


Рис. 5.4 ❖ Генерация энтропии и шифрование в виде кода восстановления

В табл. 5.4 представлена зависимость между размером данных энтропии и длиной кода восстановления в словах.

Таблица 5.4. BIP39: энтропия и длина слова

Энтропия (бит)	Контрольная сумма (бит)	Энтропия + контрольная сумма (бит)	Кодовые слова восстановления
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

От кода восстановления к seed-числу

Код восстановления представляет собой энтропию размером от 128 до 256 бит. Далее энтропия служит для получения более длинного (512-битного) seed-числа с помощью функции растяжения ключей seed (key-stretching function) PBKDF2 (https://ru.wikipedia.org/wiki/PBKDF2). После этого полученное начальное seed-число используется для создания детерминированного кошелька и извлечения из него ключей.

Функция растяжения ключа получает два параметра: энтропию и так называемую «*соль*» (*salt*). Назначение соли (модификатора) в функции растяжения ключа – усложнить создание таблицы поиска, позволяющей осуществить атаку методом перебора всех вариантов. В стандарте BIP39 соль имеет еще одно назначение – она дает возможность ввести парольную фразу, которая служит дополнительным фактором безопасности для защиты начального (seed) числа-ключа, о котором более подробно рассказано в разделе «Опциональная парольная фраза в BIP39».

Функция растягивания ключа с ее 2048 циклами хеширования делает несколько более сложной атаку методом перебора вариантов кода восстановления с помощью программного обеспечения. На аппаратные средства специального назначения влияние этой функции незначительно. Для злоумышленника, который пытается угадать весь пользовательский код восстановления, длина кода (как минимум 128 бит) обеспечивает более чем достаточную защиту. Но для тех случаев, когда взломщик может знать небольшую часть этого кода, растягивание ключа повышает безопасность, поскольку замедляет скорость проверки различных комбинаций кода восстановления. Параметры BIP39 считались слабыми по современным меркам уже при первой его публикации почти десять лет назад, хотя, возможно, это является следствием его разработки для совместимости с аппаратными устройствами подписи на маломощных процессорах. Некоторые альтернативы BIP39 используют более мощные параметры растяжения ключа, например фраза 32 768 циклов хеширования в Aezeed с использованием более сложного алгоритма Scrypt, хотя они могут быть не столь удобны для работы на аппаратных устройствах подписи.

Процесс, описанный далее в шагах с 7 по 9, продолжает описанный ранее в разделе «Генерация кода восстановления».

7. Первым параметром функции растяжения ключа PBKDF2 является энтропия, полученная на шаге 6.
8. Вторым параметром функции растягивания ключа PBKDF2 является соль. Она состоит из строки-константы «mnemonic», объединенной с опциональной строкой пользовательской парольной фразы.
9. Растягивание кода восстановления и параметров соли PBKDF2 выполняется с помощью 2048 циклов хеширования по алгоритму HMAC-SHA512, в результате чего на выходе получается 512-битное значение. Это 512-битное значение является seed-числом.

На рис. 5.5 представлено использование кода восстановления для генерации seed-числа.

В табл. 5.5–5.7 приведены примеры кодов восстановления и создаваемых ими seed-чисел.

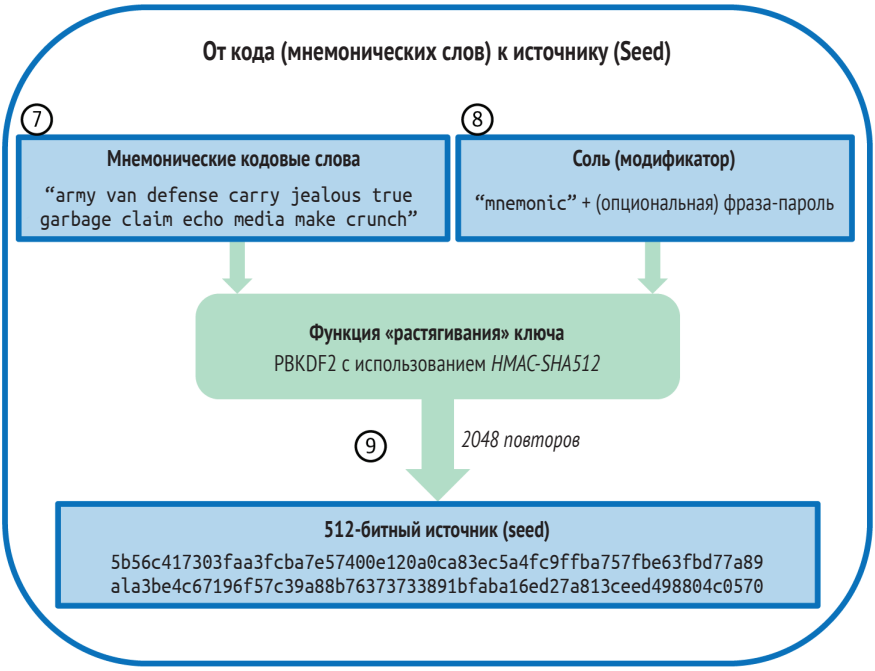


Рис. 5.5 ❖ От кода восстановления к seed-числу

Таблица 5.5. Код восстановления BIP39 со 128-битной энтропией, без парольной фразы, полученное seed-число

Входная энтропия (128 бит)	`0c1e24e5917779d297e14d45f14e1a1a`
Код восстановления (12 слов)	`army van defense carry jealous true garbage claim echo media make crunch`
Парольная фраза	Нет
Seed-число (512 бит)	`5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570`

Таблица 5.6. Код восстановления BIP39 со 128-битной энтропией, с парольной фразой, полученное seed-число

Входная энтропия (128 бит)	`0c1e24e5917779d297e14d45f14e1a1a`
Код восстановления (12 слов)	`army van defense carry jealous true garbage claim echo media make crunch`
Парольная фраза	SuperDuperSecret
Seed-число (512 бит)	`3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0715861dc8a18358f80b79d49acf64142ae57037d1d54`

Таблица 5.7. Код восстановления BIP39 с 256-битной энтропией, без парольной фразы, полученное seed-число

Входная энтропия (128 бит)	`2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c`
Код восстановления (12 слов)	`cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige`

Таблица 5.7 *(окончание)*

Парольная фраза	Нет
Seed-число (512 бит)	3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3 a04356e53d062e55f1e0deaa082df8d487381379df848a6ad7e98798404

Сколько нужно энтропии?

В BIP32 допускается размер seed-чисел от 128 до 512 бит. Для BIP39 допустимо от 128 до 256 бит; для Electrum v2 – 132 бита; для Aezeed – 128 бит; для SLIP39 – 128 или 256 бит энтропии. Разброс этих значений оставляет неясным, сколько же энтропии требуется для обеспечения безопасности. Попробуем прояснить этот вопрос.

Расширенные секретные ключи BIP32 состоят из 256-битного ключа и 256-битного кода цепочки, что в сумме составляет 512 бит. Это означает, что существует максимум 2^{512} различных вариантов расширенных секретных ключей. Если начать с более чем 512 бит энтропии, все равно выйдет расширенный секретный ключ с 512 битами энтропии, поэтому нет смысла использовать более 512 бит, даже если бы любой из упомянутых выше стандартов это позволял.

Несмотря на наличие 2^{512} различных расширенных секретных ключей, обычных секретных ключей всего 2^{256} (чуть меньше), и именно эти секретные ключи обеспечивают безопасность ваших биткойнов. То есть даже если использовать более 256 бит энтропии для исходного seed-числа, все равно получится секретный ключ с 256 битами энтропии. Не исключено, что в будущем появятся биткойн-протоколы с дополнительной энтропией в расширенных ключах для повышения безопасности, но сейчас такого нет.

Уровень защиты открытого ключа Биткойна составляет 128 бит. Взломщик с классическим компьютером (на сегодня единственный вид систем, который применим для практической атаки) должен выполнить порядка 2^{128} операций на эллиптической кривой Биткойна для подбора секретного ключа к открытому ключу другого пользователя. Из этого следует, что использование 128 бит энтропии не дает видимых преимуществ (хотя при этом необходимо обеспечить равномерный выбор из всего диапазона 2^{256} секретных ключей).

У большей энтропии есть одно дополнительное преимущество: если взломщик знает определенный процент вашего кода восстановления (но не весь код), то чем больше энтропия, тем сложнее ему будет вычислить неизвестную ему часть кода. Например, если злоумышленник получит половину 128-битного кода (64 бита), вполне вероятно, что он сможет перебрать оставшиеся 64 бита. Если же он знает половину 256-битного кода (128 бит), то вероятность перебора оставшейся половины очень мала. Полагаться на эту защиту мы не рекомендуем – нужно либо хранить коды восстановления в максимальной безопасности, либо использовать методы вроде SLIP39, которые позволяют распределить код восстановления по нескольким местам, не полагаясь на безопасность каждого отдельного кода.

По состоянию на 2023 год большинство современных кошельков генерируют 128 бит энтропии для своих кодов восстановления (или значение, близкое к 128, как, например, 132 бита в Electrum v2).

Опциональная парольная фраза в BIP39

Стандарт BIP39 допускает использование опциональной парольной фразы при получении seed-числа. Если парольная фраза не используется, код восстановления растягивается с солью, которая включает в себя строчную константу «mnemonic». В результате из любого заданного кода восстановления получается определенное 512-битное исходное seed-число. При использовании парольной фразы функция растягивания производит другое seed-число из того же кода восстановления. По сути, из одного кода восстановления каждая возможная парольная фраза позволяет получить разные seed-числа. Таким образом, не существует «неправильной» парольной фразы. Все они действительны, и все они приводят к различным исходным числам, образуя огромное множество потенциальных неинициализированных кошельков. Набор всех возможных кошельков настолько велик (2^{512}), что нет никакой практической возможности перебора или случайного угадывания используемого кошелька.

В BIP39 не существует «неправильных» кодовых фраз. Каждая парольная фраза ведет к некоторому кошельку. Если кошелек ранее не использовался, он будет пуст.

Дополнительная парольная фраза выполняет две важные функции:

- второй фактор (что-то запоминаемое), который делает код восстановления бесполезным сам по себе, обеспечивая таким образом защиту кодов восстановления от взлома случайным вором. Для защиты от технически подкованных похитителей необходимо использовать очень сильную парольную фразу;
- форма правдоподобного отрицания, или «кошелек под принуждением», когда выбранная парольная фраза ведет к кошельку с небольшим количеством средств для отвлечения злоумышленника от «настоящего» кошелька с большей частью денег.

Важно отметить, что при использовании парольной фразы также возникает риск потери:

- если владелец кошелька недееспособен или умер и никто больше не знает кодовую фразу, исходный ключ (seed) окажется бесполезным, а все хранящиеся в кошельке средства будут потеряны навсегда;
- и наоборот, если владелец сохранит кодовую фразу в том же месте, что и seed-число, это лишит второй фактор всякого смысла.

Несмотря на значительную пользу парольных фраз, их следует использовать только в сочетании с детально продуманным методом резервного копирования и восстановления, с учетом перспективы выживания владельца и возможности его или ее семьи вернуть наследство в криптовалюте.

Создание HD-кошелька из seed-числа

HD-кошельки (Hierarchical Deterministic wallets – иерархические детерминированные кошельки) создаются из одного корневого seed-числа, которым является 128-, 256- или 512-битное случайное число. Чаще всего это число генери-

руется или расшифровывается с помощью кода восстановления, как подробно описано в предыдущем разделе.

Каждый ключ в HD-кошельке детерминированно получен из этого исходного seed-числа, что дает возможность воссоздать весь HD-кошелек из него в любом совместимом HD-кошельке. Поэтому можно легко создавать резервные копии, восстанавливать, экспортировать и импортировать HD-кошельки, содержащие тысячи или даже миллионы ключей, передавая лишь код восстановления, полученный из корневого seed-числа. Процесс создания мастер-ключей и мастер-кода цепочки для HD-кошелька показан на рис. 5.6.

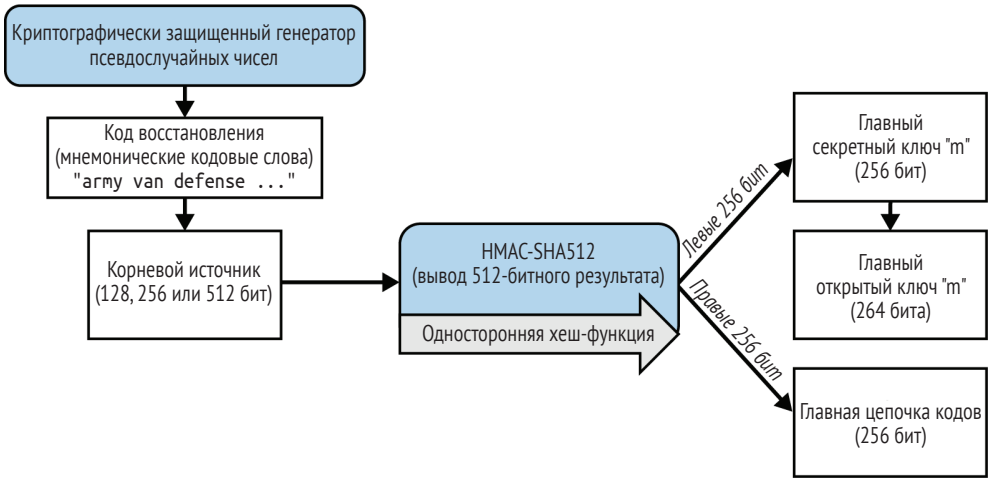


Рис. 5.6 ❖ Создание мастер-ключей и кода цепочки из корневого seed-числа

Корневое seed-число вводится в алгоритм HMAC-SHA512, полученный хеш используется для создания *главного секретного ключа* (*master private key – m*) и *главной цепочки кодов* (*master chain code – c*).

Затем из главного секретного ключа (*m*) с помощью обычного умножения на эллиптической кривой $m \times G$ генерируется соответствующий главный открытый ключ (*M*), который уже был рассмотрен в разделе «Открытые ключи».

Код мастер-цепочки (*c*) используется для внесения энтропии в функцию создания дочерних ключей из родительских, как будет показано в следующем разделе.

Деривация секретного дочернего ключа

HD-кошельки используют функцию *деривации* (выведения) *дочерних ключей* (*Child Key Derivation*, *CKD*) для получения дочерних ключей из родительских.

Функции получения дочерних ключей построены на принципах работы односторонней хеш-функции, которая объединяет:

- родительский секретный или открытый ключ (несжатый ключ);
- seed-число, называемое кодом цепочки (chain code, 256 бит);
- номер индекса (32 бита).

Код цепочки (chain code) используется для введения в процесс детерминированных случайных данных, поэтому для получения других дочерних ключей недостаточно знать индекс и дочерний ключ. Кроме того, знание дочернего ключа не позволяет найти его «братьев и сестер», если не знать код цепочки. Начальный код цепочки (в корне дерева) создается из seed-кода, а последующие дочерние коды цепочки получаются из каждого родительского кода цепочки.

Эти три элемента (родительский ключ, код цепочки и индекс) объединяются и хешируются для создания дочерних ключей следующим образом.

Родительский открытый ключ, код цепочки и индекс объединяются и хешируются алгоритмом HMAC-SHA512 для получения 512-битного хеша. Этот 512-битный хеш разбивается на две 256-битные половинки. Правые 256 бит выходного хеша становятся дочерним кодом цепочки. Левая половина 256 бит хеша добавляется к родительскому секретному ключу для создания дочернего секретного ключа. На рис. 5.7 показан пример с индексом 0 для получения «нулевого» (первого по индексу) дочернего ключа от родительского.

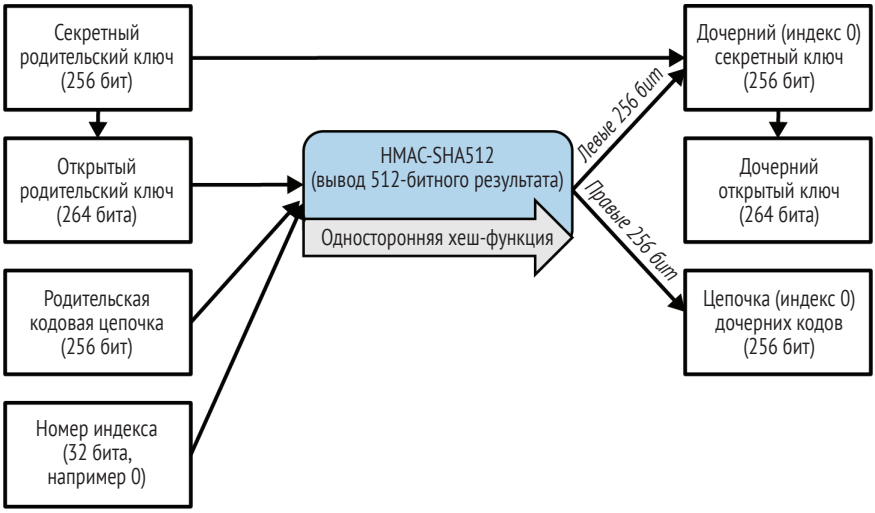


Рис. 5.7 ❖ Увеличение родительского секретного ключа для создания дочернего секретного ключа

Изменение индекса позволит расширить родительский ключ и создать другие дочерние ключи в последовательности (например, Child 0, Child 1, Child 2 и т. д.). Каждый родительский ключ может иметь 2 147 483 647 (2^{31}) дочерних ключей (2^{31} – это половина всего доступного диапазона 2^{32}: другая половина зарезервирована для особого типа деривации, о котором будет рассказано далее в этой главе).

Повторяя процесс на уровень ниже по дереву, каждый дочерний ключ может становиться родительским и образовывать свои собственные дочерние ключи в бесконечном количестве поколений.

Использование производных дочерних ключей

Дочерние секретные ключи неотличимы от недетерминированных (случайно сгенерированных) ключей. Поскольку функция деривации (выведения) является односторонней, дочерний ключ не удастся применить для поиска родительского ключа. Дочерний ключ также не поможет найти «братьев и сестер». Если имеется n-й дочерний ключ, найти его «родственников», например $n - 1$ или $n + 1$, или любые другие дочерние ключи, входящие в эту последовательность, невозможно. Только родительский ключ и код цепочки позволяют вывести все дочерние ключи. Без кода цепочки дочерний ключ также не может быть использован для получения «внуков». Чтобы начать новую ветвь и произвести ключи уровня «внуков», потребуются секретный дочерний ключ и код дочерней цепочки.

Для чего же можно использовать дочерний секретный ключ? Для создания открытого ключа и адреса Биткойна. Затем его можно применять для подписания транзакций по расходованию перечисляемых на этот адрес средств.

Дочерний секретный ключ, соответствующий открытый ключ и биткойн-адрес неотличимы от ключей и адресов, созданных случайным образом. Тот факт, что они являются частью последовательности, не виден за пределами создавшей их функции HD-кошелька. После создания они работают точно так же, как и «обычные» ключи.

Расширенные ключи

Как уже говорилось, функцию извлечения (деривации) ключа (key derivation function) можно использовать для создания дочерних ключей на любом уровне дерева при наличии трех входных данных: ключа, кода цепочки и индекса нужного дочернего ключа. Два основных компонента – это ключ и код цепочки, которые вместе называют *расширенным ключом* (*extended key*). Термин «расширенный ключ» также можно трактовать как «*расширяемый ключ*» (*extensible key*), поскольку с его помощью можно вывести дочерние ключи.

Расширенные ключи сохраняются и отображаются в виде простого сочетания ключа и кода цепочки. Существует два типа расширенных ключей. Расширенный секретный ключ, состоящий из секретного ключа и кода цепочки, может использоваться для создания дочерних секретных ключей (а на их основе – дочерних открытых ключей). Расширенный открытый ключ, включающий в себя открытый ключ и код цепочки, может использоваться для создания дочерних открытых ключей (только открытых), как описано в разделе «Открытые ключи».

Расширенный ключ можно рассматривать как корень ветви в древовидной структуре HD-кошелька. Имея корень ветви, можно получить все остальные ветви. Расширенный секретный ключ может образовать полную ветвь, в то время как расширенный открытый ключ – только ветвь открытых ключей.

Расширенные ключи кодируются с помощью base58check, что упрощает экспорт и импорт между различными BIP32-совместимыми кошельками. В кодировке base58check для расширенных ключей используется специальный номер версии, который при кодировке в символах base58 дает префикс «xprv»

и «xpub», чтобы сделать их легкоузнаваемыми. Поскольку расширенный ключ содержит гораздо больше байтов, чем обычные адреса, он также намного длиннее других ранее рассмотренных строк с кодировкой base58check.

Вот пример расширенного секретного ключа, закодированного в base-58check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CA
WrUE9i6GoNMKUga5biW6Hx4tws2six3b9c
```

Вот соответствующий расширенный открытый ключ, закодированный в base58check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBP
LrtJunSDMstweyLXhRgPxdp14sk9tJPW9
```

Получение открытых дочерних ключей

Как уже упоминалось, очень полезной характеристикой HD-кошельков является возможность получения открытых дочерних ключей из открытых родительских ключей без использования секретных ключей. Это позволяет получить дочерний открытый ключ двумя способами: либо из дочернего секретного ключа, либо непосредственно из родительского открытого ключа.

Поэтому расширенный открытый ключ можно использовать для получения всех открытых ключей (но только открытых) в этой ветви структуры HD-кошелька.

Этот прием можно использовать для установки только открытых ключей, когда у сервера или приложения есть копия расширенного открытого ключа и нет секретных ключей. Такой тип установки может создавать бесконечное количество открытых ключей и адресов Биткойна, но не может тратить деньги, отправленные на эти адреса. При этом на другом, более защищенном сервере из расширенного секретного ключа могут быть получены все соответствующие секретные ключи для подписания транзакций и расходования денег.

Одним из наиболее популярных вариантов применения этого решения является установка расширенного открытого ключа на веб-сервер для приложений электронной коммерции. Веб-сервер может использовать функцию деривации открытого ключа и создавать новый биткойн-адрес для каждой транзакции (например, для корзины покупателя). При этом у веб-сервера не будет секретных ключей, которые можно было бы украсть. В отсутствие HD-кошельков единственным способом сделать это будет генерация тысяч биткойн-адресов на отдельном защищенном сервере, с последующей их предварительной загрузкой на сервер электронной коммерции. Такой подход громоздок и нуждается в постоянном обслуживании, чтобы гарантировать постоянное наличие ключей на этом сервере.

Не забывайте о разрывах

Расширенный открытый ключ может создать около 4 млрд прямых дочерних ключей, что гораздо больше, чем нужно любому магазину или приложению. Однако создание всех 4 млрд ключей и сканирование блокчейна на предмет транзакций,

связанных с этими ключами, также займет у приложения кошелька неоправданно много времени. По этой причине большинство кошельков одновременно генерируют только несколько ключей и сканируют блокчейн на предмет платежей с использованием этих ключей, а дополнительные ключи генерируют в той последовательности, в которой использовались предыдущие. Например, кошелек Алисы генерирует 100 ключей. Когда она видит платеж по первому ключу, она генерирует 101-й ключ.

Иногда приложение кошелька передает ключ кому-нибудь, кто позже решает отказаться от оплаты, создавая разрыв (gap) в цепочке ключей. Это нормально, если кошелек уже создал ключи после разрыва, поэтому он найдет последующие платежи и продолжит генерировать новые ключи. Максимальное количество неиспользованных ключей подряд, которые могут не получить платеж без возникновения проблем, называется *лимитом разрыва* (*gap limit*).

Когда приложение кошелька распределило все ключи до своего лимита разрыва и ни один из этих ключей не получил выплату, у него есть три варианта обработки будущих запросов на новые ключи:

1) он может отклонять запросы, препятствуя получению дальнейших платежей. Очевидно, что это неприемлемый вариант, хотя и самый простой в реализации;
2) он может генерировать новые ключи сверх своего лимита пробелов. Это гарантирует получение уникального ключа для каждого, кто запрашивает платеж, что предотвращает повторное использование адресов и повышает конфиденциальность. Однако при попытке восстановить кошелек с помощью кода восстановления или если владелец кошелька использует другое программное обеспечение с тем же расширенным открытым ключом, другие кошельки не смогут увидеть платежи, полученные после увеличения лимита разрыва;
3) он может раздавать ранее распределенные ключи для плавного восстановления, но потенциально снижает конфиденциальность владельца кошелька и людей, с которыми он совершает сделки.

Системы с открытым исходным кодом для онлайн-продавцов, такие как BTCPay Server, стараются обойти эту проблему с помощью очень больших лимитов разрыва и ограничения скорости генерации счетов. Были предложены и другие решения, например запрос кошельку отправителя на создание (но не передачу) транзакции с оплатой вероятно повторно используемого адреса до того, как он получит новый адрес для проведения фактической транзакции. Однако на момент написания книги эти другие решения еще не были реализованы на практике.

Еще одно типичное применение такого решения – устройства «холодного» хранения или аппаратные устройства подписи. В этом случае расширенный секретный ключ может храниться в бумажном кошельке или на аппаратном устройстве, а расширенный открытый ключ – в интернете. Пользователь может создавать адреса «получения» по своему усмотрению, в то время как секретные ключи надежно хранятся вне сети. Чтобы потратить средства, пользователь может использовать расширенный секретный ключ в приложении офлайн-кошелька или на аппаратном устройстве подписи. На рис. 5.8 показан

механизм расширения родительского открытого ключа для получения дочерних открытых ключей.

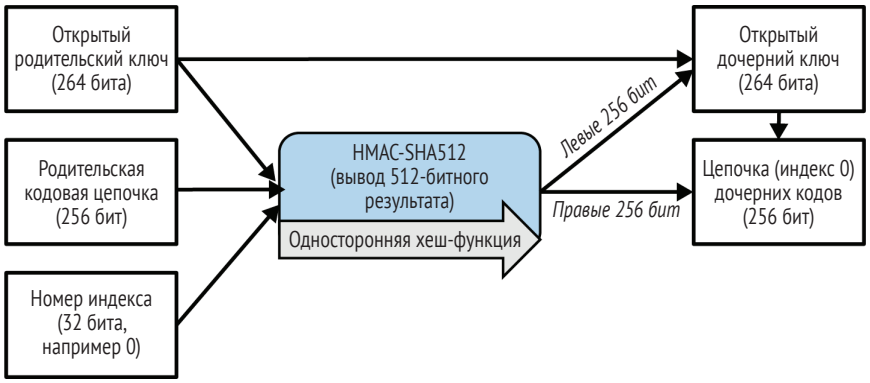


Рис. 5.8 ❖ Расширение родительского открытого ключа для создания дочернего открытого ключа

Использование расширенного открытого ключа в интернет-магазине

Использование HD-кошельков можно рассмотреть на примере интернет-магазина Габриэля.

Сначала Габриэль создал свой интернет-магазин в качестве хобби с помощью простого шаблона WordPress на хостинге. Его магазин был довольно простым: всего несколько страниц и форма заказа с одним биткойн-адресом.

Габриэль использовал первый биткойн-адрес, сгенерированный его постоянным кошельком, в качестве основного биткойн-адреса своего магазина. Клиенты оформляли заказ с помощью соответствующей формы и отправляли оплату на опубликованный биткойн-адрес Габриэля, после чего по электронной почте ему приходило письмо с деталями заказа для обработки. При небольшом количестве заказов в неделю эта система работала достаточно хорошо, хотя и снижала конфиденциальность Габриэля, его клиентов и получателей платежей.

Однако маленький интернет-магазин стал довольно успешным и привлек множество заказов от местной общины. Вскоре Габриэль оказался завален работой. Поскольку все заказы поступали на один и тот же адрес, стало сложнее правильно сопоставлять заказы и транзакции, особенно когда несколько заказов на одну и ту же сумму поступали практически одновременно.

Единственные метаданные, которые выбирает получатель типичной биткойн-транзакции, – это сумма и адрес платежа. Нет ни темы, ни поля сообщения, где можно было бы хранить уникальный идентификационный номер счета.

Для Габриэля HD-кошелек является более эффективным решением благодаря возможности получения открытых дочерних ключей без указания секретных ключей. Габриэль может загрузить расширенный открытый ключ (xpub) на свой веб-сайт. Его можно использовать при создании уникального адреса для

каждого заказа клиента. Уникальный адрес сразу повышает уровень конфиденциальности, а также присваивает каждому заказу уникальный идентификатор, по которому можно отслеживать оплату счетов.

Использование HD-кошелька дает Габриэлю возможность тратить средства из своего личного приложения-кошелька, но xpub, загруженный на сайт, может только генерировать адреса и получать средства. Эта особенность HD-кошельков является важным элементом безопасности. Сайт Габриэля не содержит секретных ключей, поэтому при любом его взломе удастся украсть только те средства, которые Габриэль мог бы получить в будущем, но никак не полученные им ранее.

Для экспорта xpub из своего устройства аппаратной подписи Trezor Габриэль использует веб-приложение кошелька Trezor. Чтобы экспортировать открытые ключи, устройство Trezor должно быть подключено к сети. Следует отметить, что большинство устройств аппаратной подписи никогда не экспортируют секретные ключи – они всегда остаются на устройстве.

Габриэль копирует xpub в программу обработки биткойн-платежей своего интернет-магазина, например на широко используемый BTCPay Server с открытым исходным кодом.

Усиленная деривация дочерних ключей

Возможность извлечения (деривации) ветви открытых ключей из xpub очень полезна, но сопряжена с потенциальным риском. Доступ к xpub не дает доступа к дочерним секретным ключам. Но поскольку xpub содержит код цепочки, то если дочерний секретный ключ известен или каким-то образом разглашен, его можно использовать вместе с кодом цепочки для получения всех остальных дочерних секретных ключей. Утечка одного дочернего секретного ключа вместе с родительским кодом цепочки раскрывает все секретные ключи всех дочерних ключей. Хуже того, дочерний секретный ключ вместе с родительским кодом цепочки может быть использован для получения родительского секретного ключа.

Чтобы противостоять этому риску, HD-кошельки предоставляют альтернативную функцию извлечения, называемую *усиленной деривацией* (*hardened derivation*), которая разрывает связь между родительским открытым ключом и дочерним кодом цепочки. Функция усиленной деривации использует родительский секретный ключ для извлечения кода дочерней цепочки, а не родительский открытый ключ. Это создает «межсетевой экран» в последовательности родительский–дочерний, с кодом цепочки, который не удастся использовать для компрометации родительского или дочернего секретного ключа. Усиленная функция деривации выглядит почти так же, как и обычная деривация дочернего секретного ключа, за исключением того, что вместо родительского открытого ключа в качестве входных данных хеш-функции используется родительский секретный ключ, как показано на рис. 5.9.

При использовании усиленной частной функции деривации полученный дочерний секретный ключ и код цепочки полностью отличаются от тех, которые получились бы при применении обычной функции деривации. Образовавшаяся «ветвь» ключей может быть использована для создания расширенных от-

крытых ключей, которые оказываются неуязвимыми, поскольку содержащийся в них код цепочки не может быть использован для получения секретных ключей их «братьев-сестер» или «родителей». Таким образом, усиленная деривация используется для создания «разрыва» в дереве над уровнем, на котором используются расширенные открытые ключи.

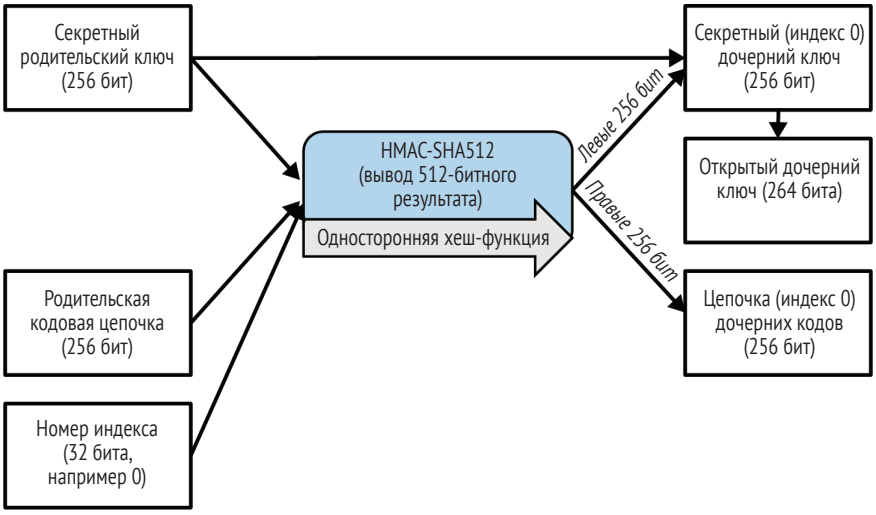


Рис. 5.9 ❖ Усиленная деривация дочернего ключа; без родительского открытого ключа

Иными словами, если нужно использовать удобство xpub для создания ветвей открытых ключей, не подвергая себя риску утечки кода цепочки, следует применять не обычный родительский ключ, а родительский усиленный. Согласно лучшим рекомендациям, дочерние элементы мастер-ключей первого уровня всегда выводятся через усиленную деривацию для предотвращения компрометации мастер-ключей.

Индексы для обычной и усиленной дериваций

Используемый в функции деривации индексный номер представляет собой 32-битное целое число. Чтобы было легче отличить ключи, созданные с помощью обычной функции деривации, от ключей, полученных с помощью усиленной деривации, этот индексный номер разделен на два диапазона. Номера индексов от 0 до $2^{31} - 1$ (от 0x0 до 0x7FFFFFF) используются *только* для обычной деривации. Номера индексов от 2^{31} до $2^{32} - 1$ (от 0x80000000 до 0xFFFFFF) используются *только* для усиленной деривации. Таким образом, если номер индекса меньше 2^{31}, дочерний ключ является обычным, а если номер индекса равен или больше 2^{31}, дочерний ключ является усиленным.

Для более удобного чтения и отображения номер индекса для усиленных дочерних ключей начинается с нуля, но с символом «штрих» (prime symbol), то есть «'». Таким образом, первый обычный дочерний ключ отображается как 0,

в то время как первый усиленный дочерний ключ (индекс 0x80000000) отображается как 0'. Затем в последовательности второй усиленный ключ будет иметь индекс 0x80000001 и будет отображаться как 1' и т. д. Когда в HD-кошельке отображается индекс i', это означает 2^{31}+i. В обычной текстовой системе ASCII символ «штрих» (prime) заменяется либо одиночным апострофом, либо буквой *h*. В таких ситуациях, как дескрипторы скриптов выхода, где текст может использоваться в оболочке или другом контексте, где одиночный апостроф имеет особое значение, рекомендуется использовать букву *h*.

Идентификатор ключа HD-кошелька (путь)

Идентификация ключей в HD-кошельке происходит в соответствии с соглашением «пути» (path), где каждый уровень дерева разделяется символом косой черты (/) (см. табл. 5.8). Секретные ключи, полученные от главного секретного ключа, начинаются с буквы «m». Открытые ключи, полученные от главного открытого ключа, начинаются с «M». Таким образом, первый дочерний секретный ключ главного секретного ключа обозначается m/0. Первый дочерний открытый ключ обозначается M/0. Второй потомок первого дочернего ключа («внук») обозначается m/0/1 и т. д.

«Родословная» ключа читается справа налево, пока не достигается главный ключ, от которого он был получен. Например, идентификатор m/x/y/z описывает ключ, который является Z-потомком ключа m/x/y, который является Y-потомком ключа m/x, который является x-потомком m.

Таблица 5.8. Примеры путей к HD-кошелькам

HD-путь	Описание ключа
m/0	Первый (0) дочерний секретный ключ от главного секретного ключа (m)
m/0/0	Первый секретный ключ «внука» от первого дочернего ключа (m/0)
m/0'/0	Первый обычный секретный ключ «внука» от первого усиленного дочернего ключа (m/0')
m/1/0	Первый секретный ключ «внука» от второго дочернего ключа (m/1)
M/23/17/0/0	Первый открытый ключ «прапраправнука» от первого «праправнука» от 18-го «внука» от 24-го дочернего ключа

Навигация по древовидной структуре HD-кошелька

Древовидная структура HD-кошелька обладает невероятной гибкостью. У каждого родительского расширенного ключа может быть 4 млрд дочерних: 2 млрд обычных и 2 млрд усиленных. Каждый из этих дочерних ключей может иметь еще 4 млрд дочерних ключей и т. д. Дерево может быть сколь угодно разветвленным, с бесконечным числом поколений. Однако при всей этой гибкости ориентироваться в бесконечном дереве становится довольно сложно. Особенно сложно переносить HD-кошельки между разными вариантами реализации, поскольку возможности внутренней организации ветвей и ответвлений бесконечны.

Два протокола BIP дают возможность решить эту проблему за счет введения некоторых предлагаемых стандартов для структур деревьев HD-кошельков.

В BIP43 предлагается использовать индекс первой дочерней ветви в качестве специального идентификатора для обозначения «предназначения» (purpose) структуры дерева. Согласно BIP43, HD-кошелек должен использовать только одну ветвь дерева первого уровня, при этом номер индекса определяет структуру и пространство имен остального дерева через определение его назначения. Например, HD-кошелек, использующий только ветвь m/i'/, предназначен для обозначения конкретного назначения, и это назначение обозначается индексом «i».

Дополняя эту спецификацию, в BIP44 предлагается мультиаккаунтная структура в качестве «назначения» под номером 44' в рамках BIP43. Все HD-кошельки, соответствующие структуре BIP44, идентифицируются по факту использования только одной ветви дерева: m/44'/.

В BIP44 структура определяется пятью предопределенными уровнями дерева:

```
m / purpose' / coin_type' / account' / change / address_index
```

Первый уровень «назначения» всегда имеет значение 44'. Второй уровень «coin_type» определяет тип криптовалютной монеты, что позволяет создавать мультивалютные HD-кошельки с каждой валютой в отдельной части дерева («поддерева» – subsree) второго уровня. Так, например, Bitcoin – это m/44'/0', а Bitcoin Testnet – m/44'/1'.

Третий уровень дерева – «аккаунт» (account), дает возможность пользователям разделить свои кошельки на отдельные логические «субсчета» (subaccounts) для бухгалтерских или организационных целей. Например, HD-кошелек может содержать два «счета» биткойна: m/44'/0'/0' и m/44'/0'/1'. Каждый счет является корнем своего собственного поддерева.

На четвертом уровне «change» («сдача») HD-кошелек имеет два поддерева: одно для создания принимающих адресов и одно для создания адресов для сдачи. Нужно отметить, что если на предыдущих уровнях использовалась усиленная деривация, то на этом уровне используется обычная деривация. Это сделано для поддержки возможности экспорта расширенных открытых ключей на данном уровне дерева для использования в незащищенной среде. Используемые адреса выводятся HD-кошельком как дочерние элементы четвертого уровня, в результате чего пятый уровень дерева становится «адресным индексом» («address_index»). Например, третьим адресом получения платежей на основной счет будет M/44'/0'/0'/0/2. В табл. 5.9 приведено еще несколько примеров.

Таблица 5.9. Примеры структуры HD-кошелька BIP44

HD-путь	Описание ключа
M/44'/0'/0'/0/2	Третий открытый ключ получателя для основного аккаунта Биткойна
M/44'/0'/3'/1/14	Пятнадцатый открытый ключ для адреса сдачи четвертого аккаунта Биткойна
m/44'/2'/0'/0/1	Второй секретный ключ в основном аккаунте Litecoin для подписания транзакций

Большинство уделяет особое внимание защите своих биткойнов от кражи и других угроз, но одной из основных причин потери биткойнов, а возможно

и главной, является потеря данных. Если ключи и другие важные данные, необходимые для расходования биткойнов, будут утеряны, эти биткойны навсегда останутся неизрасходованными. Вернуть их вам никто не сможет. В этой главе были рассмотрены системы, с помощью которых современные приложения для работы с кошельками помогают предотвратить потерю этих данных. Однако следует помнить, что практическое использование имеющихся систем для создания хороших резервных копий и их регулярной проверки зависит только от вас.

Глава 6

Транзакции

Традиционные способы передачи физических денег имеют мало общего с порядком обращения биткойнов. Физические деньги – это деньги на предъявителя. Алиса платит Бобу, передавая ему некоторое количество денежных знаков, например долларовые купюры. В отличие от них, биткойны не существуют ни физически, ни в виде цифровых данных – Алиса не может передать Бобу несколько биткойнов или отправить их по электронной почте.

Однако можно представить, как Алиса может передать Бобу контроль над земельным участком. Она не имеет возможности физически взять землю и передать ее Бобу. Вместо этого существует какая-то запись (обычно ее ведет местный орган власти), в которой содержится описание принадлежащей Алисе земли. Она передает эту землю Бобу после получения разрешения от властей на обновление записи, в которой будут записаны права Боба на эту землю.

Подобным образом работает и Биткойн. На каждом полном узле Биткойна существует база данных, в которой записано, что Алиса контролирует определенное количество биткойнов. Если Алиса платит Бобу, она говорит полным узлам, что часть ее биткойнов теперь контролирует Боб. Информация, которую Алиса передает, чтобы побудить полные узлы обновить свои базы данных, называется *транзакцией* (*transaction*). Как будет показано в главе 7, это происходит без прямого указания личности Алисы или Боба.

В этой главе на примере биткойн-транзакции будет сделан анализ каждой из ее частей, чтобы выяснить, как они обеспечивают передачу активов с высокой эффективностью и невероятной надежностью.

Сериализованная транзакция Биткойна

В разделе «Исследование и декодирование транзакций» для получения копии платежа от Алисы к Бобу использовалось Bitcoin Core с включенной опцией txindex. Вернемся к транзакции с этим платежом, как показано в примере 6.1.

Пример 6.1 ❖ Сериализованная транзакция Алисы

```
$ bitcoin-cli getrawtransaction 466200308696215bbc949d5141a49a41\
38ecdfdfaa2a8029c1f9bcecd1f96177

01000000000101eb3ae38f27191aa5f3850dc9cad00492b88b72404f9da13569
8679268041c54a0100000000ffffffff02204e0000000000002251203b41daba
```

```
4c9ace578369740f15e5ec880c28279ee7f51b07dca69c7061e07068f8240100
00000000001600147752c165ea7be772b2c0acb7f4d6047ae6f4768e0141cf5efe
2d8ef13ed0af21d4f4cb82422d6252d70324f6f4576b727b7d918e521c00b51b
e739df2f899c49dc267c0ad280aca6dab0d2fa2b42a45182fc83e81713010000
0000
```

Формат сериализации Bitcoin Core является особенным, поскольку именно он используется для принятия обязательств по транзакциям и передачи их по P2P-сети Биткойна. Но в остальном программы могут использовать другой формат, если они передают одни и те же данные. При этом формат Bitcoin Core весьма компактен для передаваемых данных и прост для парсинга, поэтому многие другие программы Bitcoin также используют этот формат.

Еще один широко используемый формат сериализации транзакций, о котором известно, – это формат частично подписанных транзакций Биткойна (partially signed bitcoin transaction, PSBT), задокументированный в BIP 174 и BIP 370 (с расширениями, задокументированными в других BIP). Формат PSBT позволяет ненадежной программе создать шаблон транзакции, который может быть проверен и обновлен доверенными программами (например, аппаратными устройствами подписи) с секретными ключами или другими конфиденциальными данными, необходимыми для заполнения шаблона. Для этого PSBT обеспечивает хранение значительного количества метаданных о транзакции, что делает его значительно менее компактным, чем стандартный формат сериализации. В этой книге нет подробного описания PSBT, но разработчикам кошельков, которые планируют поддерживать подписание несколькими ключами, его использование настоятельно рекомендуется.

Транзакция, отображаемая в шестнадцатеричном виде в примере 6.1, показана в виде карты байтов на рис. 6.1. Обратите внимание, что для отображения 32 байт требуется 64 шестнадцатеричных символа. На этой карте показаны только поля верхнего уровня. Далее будет рассмотрено каждое из них в порядке их появления в транзакции. Также будут описаны все дополнительные поля, которые они содержат.

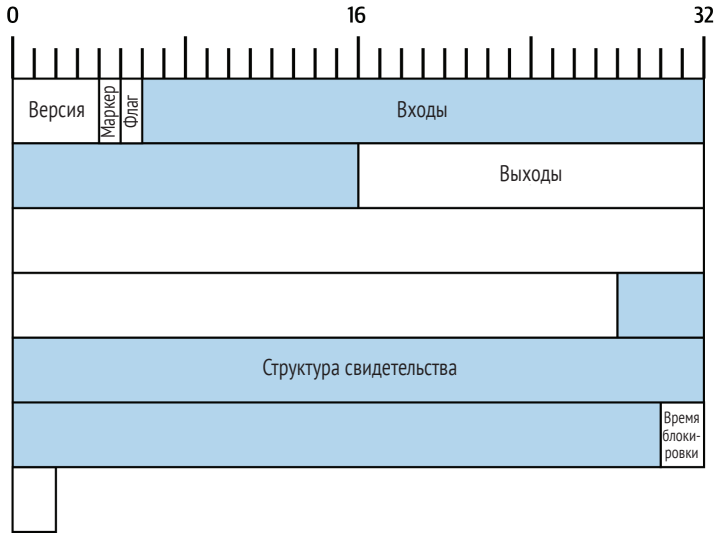


Рис. 6.1 ❖ Карта байтов транзакции Алисы

Версия

Первые четыре байта сериализованной биткойн-транзакции – это ее версия. Первоначальной версией биткойн-транзакций была версия 1 (0x01000000). Все транзакции в Биткойне должны следовать правилам транзакций версии 1, многие из которых описаны в этой книге.

Биткойн-транзакции версии 2 появились по причине изменения правил консенсуса Биткойна в результате программного ответвления (софт-форка) BIP68. Протокол BIP68 накладывает дополнительные ограничения на поле последовательности, но эти требования относятся только к транзакциям версии 2 и выше. Транзакции версии 1 не затрагиваются. В BIP112, который стал частью того же софт-форка, что и BIP68, обновился код операции (`OP_CHECKSEQUENCEVERIFY`), который больше не будет выполняться, если будет оцениваться как часть транзакции с версией меньше 2. За исключением этих двух изменений, транзакции версии 2 идентичны транзакциям версии 1.

Защита предварительно подписанных транзакций

Последним шагом перед трансляцией транзакции в сеть для включения в блокчейн является ее подписание. Однако можно подписать транзакцию без ее немедленной трансляции. Можно хранить эту предварительно подписанную транзакцию месяцами или годами, рассчитывая на возможность ее добавления в блокчейн позже, после ее передачи в сеть. За это время можно даже потерять доступ к секретному ключу (или ключам), необходимому для подписания альтернативной транзакции для расходования этих средств. И это не просто теория: несколько протоколов, построенных на основе Биткойна, включая Lightning Network, работают с предварительно подписанными транзакциями.

Для разработчиков протоколов возникает проблема, когда они помогают пользователям обновить протокол консенсуса Биткойна. Добавление новых ограничений, как это было сделано в BIP68 в отношении поля последовательности, может аннулировать некоторые предварительно подписанные транзакции. Если нет возможности создать новую подпись для эквивалентной транзакции, то деньги, проведенные в предварительно подписанной транзакции, будут потеряны навсегда.

Эта проблема решается с помощью резервирования некоторых функций транзакций для обновления, таких, например, как номера версий. Все, кто создавал предварительно подписанные транзакции до появления BIP68, должны были пользоваться транзакциями версии 1, поэтому только применение дополнительных ограничений BIP68 в отношении последовательности к транзакциям версии 2 или выше не должно приводить к отмене всех предварительно подписанных транзакций.

В случае реализации протокола, использующего предварительно подписанные транзакции, необходимо убедиться, что он не использует никаких функций, зарезервированных для будущих обновлений. Политика передачи транзакций Bitcoin Core по умолчанию не допускает применения зарезервированных функций. Можно проверить, соответствует ли транзакция этой политике, с помощью RPC `testmempoolaccept` системы Bitcoin Core в основной сети Биткойн.

На момент написания этой книги активно рассматривалось предложение о начале использования транзакций версии 3. Это предложение не направлено на изменение правил консенсуса, а только на пересмотр правил, которыми полные узлы Биткойна руководствуются для пересылки транзакций. Согласно этому предложению, на транзакции версии 3 будут наложены дополнительные ограничения, чтобы предотвратить некоторые атаки типа «отказ в обслуживании» (DoS), о которых подробнее будет рассказано в разделе «Закрепление транзакций».

Расширенные маркер и флаг

Следующие два поля в примере сериализованной транзакции были добавлены при изменении правил консенсуса Биткойна в рамках софт-форка segregated witness (segwit). Изменения в правила были внесены на основании BIP 141 и BIP 143, однако *расширенный формат сериализации* (*extended serialization format*) был определен уже в BIP144.

Если транзакция включает в себя структуру свидетеля (которая будет рассмотрена в разделе «Структура свидетеля»), маркер должен быть нулевым (0x00), а флаг – ненулевым. В действующем протоколе P2P флаг всегда должен быть равен единице (0x01). Альтернативные флаги зарезервированы для последующих обновлений протокола.

Если транзакция не нуждается в стеке свидетелей, маркер и флаг могут отсутствовать. Это согласуется с первоначальной версией формата сериализации биткойн-транзакций, который теперь называют *унаследованной сериализацией* (*legacy serialization*). Подробнее об этом см. в разделе «Унаследованная сериализация».

В унаследованной сериализации байт маркера мог быть истолкован как количество входов (ноль). Транзакция не может иметь нулевое количество входов, поэтому маркер сигнализирует современным программам о применении расширенной сериализации. Поле флага обеспечивает аналогичный сигнал, но при этом также служит для упрощения процесса обновления формата сериализации в будущем.

Входы

Поле входа содержит несколько дополнительных полей. Для начала карта этих байтов показана на рис. 6.2.

Длина списка входных данных транзакции

Список входа транзакции начинается с целого числа, указывающего на количество входов в транзакции. Минимальное значение – единица. Максимального значения не существует, но фактическим ограничением на предельный размер транзакции считается несколько тысяч входов. Данное значение кодируется как беззнаковое целое число CompactSize.

Рис. 6.2 ❖ Карта байтов в поле входа транзакции Алисы

Беззнаковые целые числа CompactSize

В системе Биткойн беззнаковые целые числа, которые часто бывают малыми, но иногда могут иметь и большие значения, обычно кодируются с помощью типа данных CompactSize. CompactSize – это версия целого числа переменной длины, поэтому его иногда называют `var_int` или `varint` (см., например, документацию к BIP 37 и BIP 144).

В разных программах, в том числе и в различных приложениях Биткойна, используется несколько разновидностей целых чисел переменной длины. Например, в Bitcoin Core для сериализации базы данных UTXO используется тип данных `VarInts`, который отличается от compactSize. Кроме того, поле nBits в заголовке блока Биткойна кодируется с помощью пользовательского типа данных, известного как `Compact`, который не имеет никакого отношения к compactSize. Говоря о целых числах переменной длины, используемых в сериализации транзакций Биткойна и других частях протокола Bitcoin P2P, всегда нужно применять полное название compactSize.

Для значений от 0 до 252 беззнаковые целые числа compactSize идентичны типу данных `uint8_t` языка C, с которым наверняка знаком любой программист. Для остальных значений до 0xffffffffffffffffffff к числу добавляется байт, указывающий на его длину, но в остальном они выглядят как обычные беззнаковые целые числа в кодировке языка C:

Таблица 6.1. Отображение значений compactSize

Значение	Количество байтов	Формат
⩾ 0 && ⩽ 252 (0xfc)	1	uint8_t
⩾ 253 && ⩽ 0xffff	3	0xfd с последующим числом в формате uint16_t
⩾ 0x10000 && ⩽ 0xffffffff	5	0xfe с последующим числом в формате uint32_t
⩾ 0x100000000 && ⩽ 0xffffffffffffffff	9	0xff с последующим числом в формате uint64_t

Каждый вход в транзакции должен содержать три поля: поле *outpoint*, поле *input script* с префиксом длины и поле *sequence*.

В следующих разделах будут рассмотрены все эти поля. Некоторые входы также включают стек свидетелей, но он сериализуется в конце транзакции, поэтому данный вопрос будет рассмотрен позже.

Поле Outpoint

Биткойн-транзакция – это запрос к полным узлам на обновление базы данных с информацией о владении денежными средствами. Чтобы Алиса передала контроль над частью своих биткойнов Бобу, ей сначала необходимо сообщить полным узлам информацию о предыдущей транзакции, в которой она получила эти биткойны. Поскольку контроль над биткойнами передается в транзакционных выходах, Алиса *указывает* на предыдущий *выход* (*output*) с помощью поля *outpoint* (букв. «*точка выхода*»). Каждый вход должен содержать одно поле outpoint.

В поле outpoint содержится 32-байтовый txid транзакции, в результате которой Алиса получила биткойны, которые теперь собирается потратить. Этот txid соответствует внутреннему порядку байтов Биткойна для хешей; см. раздел «Внутренний порядок байтов и порядок отображения».

Поскольку транзакции могут содержать несколько выходов, Алисе также необходимо определить, какой именно выход транзакции следует идентифицировать в качестве так называемого *индекса выхода* (*output index*). Индексы выходов – это 4-байтовые беззнаковые целые числа, начинающиеся с нуля.

Когда полный узел встречает поле outpoint, он использует эту информацию для поиска выхода, на который оно ссылается. Полным узлам необходимо просматривать только более ранние транзакции в блокчейне. Например, транзакция Алисы включена в блок 774 958. Проверяющий ее транзакцию полный узел ищет только предыдущий выход, на который ссылается ее поле outpoint в этом блоке и предыдущих блоках, но не в более поздних блоках. В блоке 774 958 они будут искать только транзакции, размещенные в предыдущем блоке до транзакции Алисы, что определяется порядком расположения листьев блока дерева Меркла (см. раздел «Деревья Меркла»).

Найдя предыдущий выход, полный узел получает из него несколько важных фрагментов информации:

1) количество биткойнов, привязанных к предыдущему выходу. Все эти биткойны будут переведены в этой транзакции. В приведенном примере транзакции сумма предыдущего выхода составила 100 000 сатоши;
2) условия авторизации для этого предыдущего выхода. Здесь указаны требования для использования биткойнов, привязанных к этому предыдущему выходу;
3) для подтвержденных транзакций – высота блока, подтвердившего транзакцию, и медианное прошедшее время (median time past, MTP) для этого блока. Это необходимо для условных блокировок по времени (описано в разделе «Последовательность как условная блокировка по времени с принудительным консенсусом») и выходов транзакций coinbase (описано в разделе «Транзакции Coinbase»);
4) доказательство существования предыдущего выхода в блокчейне (или в виде известной неподтвержденной транзакции) и того, что никакая другая транзакция его не использовала. Одно из правил консенсуса Биткойна запрещает использовать любой выход более одного раза в рамках действующего блокчейна. Таково *правило против двойных расходов* (*double*

spending): Алиса не может использовать один и тот же предыдущий выход для оплаты Бобу и Кэрол в отдельных транзакциях. Две транзакции, которые пытаются использовать один и тот же предыдущий выход, называются *конфликтующими транзакциями* (*conflicting transactions*), потому что только одна из них может быть включена в действующий блокчейн.

В разное время в разных реализациях полных узлов были опробованы различные подходы к отслеживанию предыдущих выходов. В настоящее время Bitcoin Core использует решение, которое считается наиболее эффективным для сохранения всей необходимой информации при минимизации дискового пространства: он ведет базу данных, в которой хранится каждый UTXO (непотраченный входящий остаток транзакции) и важные метаданные о нем (например, высота блока подтверждения). Каждый раз, когда поступает новый блок транзакций, все отработанные им выходы удаляются из базы данных UTXO, а все созданные ими выходы добавляются в базу данных.

Внутренние и отображаемые порядки байтов

Биткойн использует выходные данные хеш-функций, называемые «*дайджест*» (*digest*), по-разному. Дайджесты служат уникальными идентификаторами для блоков и транзакций; они используются для фиксации адресов, блоков, транзакций, подписей и т. д.; дайджесты итерируются в функции доказательства работы Биткойна. В некоторых случаях хеш-дайджесты отображаются пользователям с определенным порядком байтов, но внутри системы используются в другом порядке, что приводит к путанице. Например, рассмотрим предыдущий выход txid из поля outpoint в рассматриваемой транзакции:

```
eb3ae38f27191aa5f3850dc9cad00492b88b72404f9da135698679268041c54a
```

Если попытаться использовать этот txid для извлечения транзакции с помощью Bitcoin Core, произойдет ошибка, и придется изменить порядок байтов:

```
$ bitcoin-cli getrawtransaction \
  eb3ae38f27191aa5f3850dc9cad00492b88b72404f9da135698679268041c54a
error code: -5
error message:
No such mempool or blockchain transaction.
Use gettransaction for wallet transactions.

$ echo eb3ae38f27191aa5f3850dc9cad00492b88b72404f9da135698679268041c54a \
  | fold -w2 | tac | tr -d "\n"
4ac541802679866935a19d4f40728bb89204d0cac90d85f3a51a19278fe33aeb

$ bitcoin-cli getrawtransaction \
  4ac541802679866935a19d4f40728bb89204d0cac90d85f3a51a19278fe33aeb
02000000000101c25ae90c9f3d40cc1fc509ecfd54b06e35450702...
```

Это странное явление, скорее всего, является непреднамеренным следствием решения разработчиков ранних версий программного обеспечения Биткойна (https://bitcoin.stackexchange.com/questions/116730/why-does-bitcoin-core-print-sha256-hashes-uint256-bytes-in-reverse-order). На практике это означает, что разработчики биткойн-программ должны помнить об обратном порядке байтов

в идентификаторах транзакций и блоков, которые они показывают пользователям. В этой книге для обозначения данных, которые отображаются внутри транзакций и блоков, используется термин «*порядок внутренних байтов*» (*internal byte order*). Для отображения формы, показываемой пользователям, используется термин «*порядок отображаемых байтов*» (*display byte order*). Еще один набор общеупотребимых терминов: *little-endian byte order* – «*порядок байтов для внутренней версии*» и *big-endian byte order* – «*порядок байтов для отображаемой версии*».

Поле Input Script

Поле скрипта входа (input script) представляет собой наследие устаревшего формата транзакций. В рассматриваемом примере транзакция использует нативный segwit-вывод, который не требует никаких данных в поле input script, поэтому префикс длины для input script установлен равным нулю (0x00).

Для примера скрипта входа с префиксом длины, который использует старый формат выхода, можно рассмотреть один из вариантов произвольной транзакции в самом последнем блоке на момент написания этой книги:

```
6b483045022100a6cc4e8cd0847951a71fad3bc9b14f24d44ba59d19094e0a8c
fa2580bb664b020220366060ea8203d766722ed0a02d1599b99d3c95b97dab8e
41d3e4d3fe33a5706201210369e03e2c91f0badec46c9c903d9e9edae67c167b
9ef9b550356ee791c9a40896
```

Префикс длины (length) – это целое беззнаковое число формата compactSize, указывающее на длину сериализованного поля скрипта входа. В данном случае это один байт (0x6b), указывающий на длину скрипта входа в 107 байт. Подробно о парсинге и использовании скриптов будет рассказано в главе 7.

Поле Sequence

Последние четыре байта входа – это его *порядковый* (*sequence*) номер (номер в *последовательности*). Использование и значение этого поля со временем изменились.

Изначальная замена транзакций на основе последовательности

Изначально поле sequence было предназначено для создания нескольких версий одной и той же транзакции, при этом более поздние версии замещали более ранние в качестве кандидатов на подтверждение. По номеру последовательности отслеживалась версия транзакции.

Например, Алиса и Боб решили сделать ставку на игру в карты. Они начинают с того, что каждый из них подписывает транзакцию для внесения некоторой суммы денег в скрипт выхода, требующий подписи обоих для расходования средств – так называемый «*мультиподписной скрипт*» (*multisignature script*, сокращенно *multisig*). Это называется *установочной транзакцией* (*setup transaction*). Затем они создают транзакцию для расходования этих средств:

- первая версия транзакции, с nSequence 0 (0x000000), возвращает Алисе и Бобу первоначально внесенные ими деньги. Это называется *транзакцией возврата* (*refund transaction*). В данный момент никто из них не передает транзакцию возврата. Она понадобится им только в случае возникновения проблем;
- Алиса выигрывает первый раунд карточной игры, поэтому вторая версия транзакции, с последовательностью (sequence) 1, увеличивает сумму выплат Алисе и уменьшает долю Боба. Они оба подписывают обновленную транзакцию. Им не нужно передавать эту версию транзакции, если не возникнет проблем;
- Боб выигрывает второй раунд, поэтому последовательность увеличивается до 2, доля Алисы уменьшается, а доля Боба увеличивается. Они снова подписываются, но не отправляют;
- после еще многих раундов, в которых последовательность увеличивается, средства перераспределяются, а результирующая транзакция подписывается, но не передается, они решают завершить транзакцию. Создав транзакцию с окончательным балансом средств, они устанавливают последовательность в максимальное значение (0xffffffff), завершая тем самым транзакцию. Они транслируют эту версию транзакции, она передается по сети и в конечном итоге подтверждается майнерами.

Правила замены последовательности можно увидеть в действии, если рассмотреть альтернативные сценарии:

- представим, что Алиса передает финальную транзакцию с порядковым номером 0xffffffff, а затем Боб передает одну из предыдущих транзакций, где его баланс был выше. Поскольку версия транзакции Боба имеет меньший порядковый номер, полные узлы с использованием исходного кода Биткойна не будут передавать ее майнерам, а майнеры, которые также использовали исходный код, не будут ее добывать;
- в другом сценарии представим, что Боб передает более раннюю версию транзакции за несколько секунд до передачи Алисой окончательной версии. Узлы передадут версию Боба, и майнеры попытаются добыть ее, но когда придет версия Алисы с более высоким порядковым номером, узлы также передадут ее, и майнеры с исходным кодом Биткойна будут добывать ее, а не версию Боба. Если Бобу не повезет и блок будет открыт до прихода версии Алисы, то транзакция будет подтверждена именно версией Алисы.

Этот тип протокола сегодня называют *платежным каналом* (*payment channel*). Создатель Биткойна в приписываемом ему электронном письме назвал их *высокочастотными транзакциями* (*high-frequency transactions*) и описал ряд добавленных в протокол функций для их поддержки. О некоторых из этих функций будет рассказано далее. Кроме того, также будет описано использование современных версий платежных каналов в Биткойне.

С каналами оплаты, основанными исключительно на принципах последовательности, возникали некоторые проблемы. Первая заключалась в том, что правила замены транзакции с более низкой последовательностью на транзак-

цию с более высокой были лишь вопросом программной политики. У майнеров не было прямого стимула отдавать предпочтение одной версии транзакции перед любой другой. Вторая проблема заключалась в том, что первому, кто отправил свою транзакцию, могло повезти, и ее могли подтвердить, даже если это была транзакция не самой высокой последовательности. Протокол безопасности, который в нескольких процентах случаев дает сбой из-за неудачного стечения обстоятельств, нельзя назвать слишком эффективным.

Третья проблема заключалась в возможности замены одной версии транзакции на другую неограниченное количество раз. Каждая замена дает нагрузку на пропускную способность всех ретранслирующих полных узлов в сети. Например, на момент написания этой книги в сети насчитывалось около 50 000 ретранслирующих полных узлов. Злоумышленник, создающий 1000 заменяющих транзакций в минуту по 200 байт каждая, использовал бы около 20 Кбайт своей личной пропускной способности, но около 10 Гбайт пропускной способности сети полного узла каждую минуту. Если не считать стоимости оплаты пропускной способности в 20 КБ в минуту и периодической оплаты при подтверждении транзакции, злоумышленник не будет нести никаких расходов за то огромное бремя, которое он возложил на операторов полных узлов.

Для устранения риска такой атаки в одной из ранних версий программного обеспечения Биткойна был отключен первоначальный тип замены транзакций на основе последовательности. В течение нескольких лет полные узлы Биткойна не позволяли неподтвержденной транзакции с определенным входом (на что указывает поле outpoint) быть замененной другой транзакцией с тем же входом. Однако такая ситуация не могла продолжаться вечно.

Сигнал о замене транзакций по выбору

После отключения изначальной возможности замены транзакций на основе последовательности из-за вероятности злоупотреблений было предложено другое решение: запрограммировать Bitcoin Core и другое программное обеспечение ретранслирующих полных узлов таким образом, чтобы транзакция, заплатившая более высокую ставку комиссии за транзакцию, могла заменить конфликтующую транзакцию с меньшей ставкой комиссии. Эта функция называется «*заменой за комиссию*» – *replace by fee*, или сокращенно *RBF*. Некоторые пользователи и предприятия возражали против включения поддержки функции замещения транзакций в Bitcoin Core. Поэтому был достигнут компромисс, при котором для замены вновь использовалось поле sequence.

Как указано в BIP125, неподтвержденная транзакция с любым входом, у которой последовательность установлена в значение ниже 0xfffffffe (то есть по крайней мере на 2 ниже максимального значения), сигнализирует сети, что подписавший ее желает заменить ее на конфликтующую транзакцию с более высокой ставкой комиссии. Система Bitcoin Core разрешает заменять эти неподтвержденные транзакции и по-прежнему запрещает замену других транзакций. Благодаря этому пользователи и компании, возражавшие против такой замены, могут просто игнорировать неподтвержденные транзакции с сигналом BIP125 до момента их подтверждения.

В современных политиках замены транзакций, помимо ставок комиссии и сигналов последовательности, есть и другие аспекты, которые будут рассмотрены в разделе «Замена за счет повышения комиссии».

Последовательность как условная блокировка по времени с принудительным консенсусом

В разделе «Версия» было рассказано о том, что софт-форк BIP68 добавил новое ограничение для транзакций с номерами версий 2 и выше. Это ограничение относится к полю sequence.

Входы транзакций со значениями последовательности менее 2^{31} интерпретируются как входы с условной временной блокировкой. Такая транзакция может быть включена в блокчейн только после завершения срока действия предыдущего выхода (на который ссылается поле outpoint) на величину условного времени блокировки. Например, транзакция с одним входом с условной временной блокировкой в 30 блоков может быть подтверждена только в блоке, в котором между ней и блоком, содержащим выход, находится не менее 29 блоков, проведенных в том же блокчейне. Поскольку поле sequence задается для каждого входа, транзакция может содержать любое количество входов с временной блокировкой, и все они должны иметь достаточное время выдержки, чтобы транзакция считалась действительной. Флаг отключения (disable flag) дает транзакции возможность включать как входы с условной временной блокировкой (sequence < 2^{31}), так и входы без такой блокировки (sequence ⩾ 2^{31}).

Значение последовательности указывается либо в блоках, либо в секундах. Флаг типа (type-flag) служит для отличия значений с подсчетом блоков от значений с подсчетом времени в секундах. Флаг типа устанавливается в 23-м наименее значимом бите (то есть значение 1<<22). Если флаг типа установлен, то значение последовательности трактуется как кратное 512 секундам. Если флаг типа не установлен, значение последовательности интерпретируется как количество блоков.

При использовании последовательности в качестве условной временной блокировки учитываются только 16 младших битов. После определения флагов (биты 32 и 23) значение последовательности обычно «маскируется» 16-битной маской (например, `sequence & 0x0000FFFF`). Число, кратное 512 секундам, примерно равно среднему времени между блоками, поэтому максимальная условная временная задержка как в блоках, так и в секундах из 16 бит (2^{16}) составляет чуть более одного года.

На рис. 6.3 показана бинарная схема значений последовательности согласно BIP68.

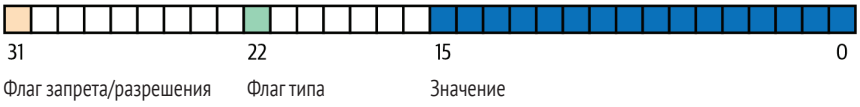


Рис. 6.3 ❖ Кодирование последовательности в соответствии с BIP68

Следует отметить, что любая транзакция, которая устанавливает условную временную блокировку с помощью последовательности, также отправляет сигнал для замены с оплатой по выбору, как описано в разделе «Сигнал о замене транзакции по выбору».

Выходы

Поле «выходы» (outputs) в транзакции состоит из нескольких полей, связанных с отдельными выходами. Как и в случае с полем входов, для начала нужно изучить определенные байты поля выходов из примера транзакции с оплатой Бобу от Алисы, которые показаны в виде карты этих байтов на рис. 6.4.

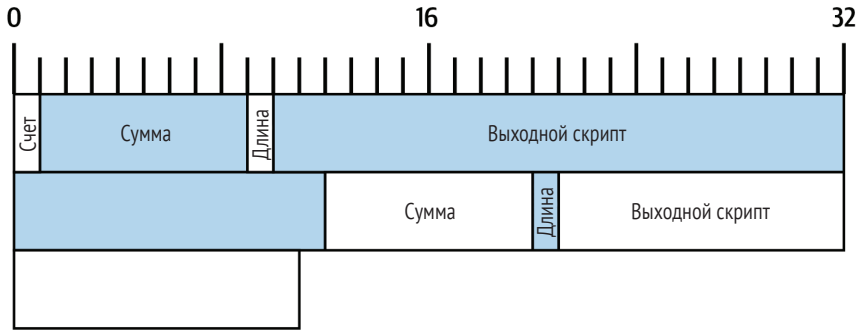


Рис. 6.4 ❖ Карта байтов поля выходов из транзакции Алисы

Количество выходов

Как и в начале секции входов транзакции, поле выходов начинается со значения count, указывающего на количество выходов в этой транзакции. Это целое число в формате compactSize, которое должно быть больше нуля.

В приведенном примере транзакции есть два выхода.

Сумма

Первое поле каждого конкретного выхода – это его *сумма* (*amount*), также обозначаемая в Bitcoin Core как «value». Это 8-байтовое целое число со знаком, указывающее количество переводимых средств в сатоши. Сатоши – это наименьшая единица биткойна, которая может быть представлена в цепочке биткойн-транзакций. В одном биткойне содержится 100 млн сатоши.

Правила консенсуса в Биткойне позволяют выводить сумму от нуля до 21 млн биткойнов (2,1 квадриллиона сатоши).

Невыгодные выходы и запрещенная пыль

Выход с нулевой стоимостью, несмотря на отсутствие какой-либо ценности, может быть потрачен по тем же правилам, что и любой другой. Однако расхо-

дование выхода (использование его в качестве входа в транзакцию) приводит к увеличению размера транзакции, что влечет за собой повышение размера комиссии. Если стоимость выхода меньше стоимости дополнительной комиссии, тратить такой выход не очень разумно с экономической точки зрения. Их называют *невыгодными выходами* (*uneconomical outputs*).

Выход с нулевой стоимостью – это всегда невыгодный выход; он не принесет никакой пользы транзакции, в которой он применяется, даже если ставка комиссии за эту транзакцию будет равна нулю. Однако многие другие выходы с низкой стоимостью также могут быть невыгодны, в том числе и случайно. Например, при типичных для сети сегодняшних ставках комиссии выход может принести транзакции больше пользы, чем стоит потратить, но завтра ставки комиссии могут вырасти, и выход станет невыгодным.

Необходимость для полных узлов отслеживать все UTXO, как описано в разделе «Поле Outpoint», означает, что каждый UTXO делает работу полного узла немного сложнее. Для UTXO со значительной стоимостью есть смысл в конечном итоге их потратить, поэтому с ними нет проблем. Но у того, кто держит под контролем невыгодные UTXO, нет никаких причин когда-либо их расходовать, и это может стать вечным бременем для операторов полных узлов. Поскольку децентрализация Биткойна зависит от желания множества людей управлять полными узлами, некоторые реализации полных узлов, такие как Bitcoin Core, препятствуют созданию невыгодных выходов с помощью правил, которые оказывают влияние на передачу и добычу неподтвержденных транзакций.

Политика против передачи или майнинга транзакций, создающих новые невыгодные выходы, получила название «политика *пыли*» (*dust* policies) за счет образного сравнения между выходами с очень маленькими суммами и частицами крохотного размера. Политика пыли в Bitcoin Core сложна и содержит ряд произвольных чисел, поэтому многие известные программы просто подразумевают, что выходы менее 546 сатоши являются пылью и по умолчанию не будут передаваться или добываться. Время от времени появляются предложения о снижении лимитов на пыль и встречные предложения об их повышении, поэтому разработчикам, использующим транзакции с предварительной подписью или многосторонние протоколы, рекомендуется проверять изменения в политике с момента публикации этой книги.

С момента создания Биткойна все полные узлы должны хранить копии всех UTXO, но это может оказаться не всегда возможным. В настоящее время ряд разработчиков занимаются проектом Utreexo, который позволяет полным узлам хранить обязательства по набору UTXO, а не сами данные. Минимальное обязательство может быть размером всего в килобайт или два. Для сравнения: в настоящее время в Bitcoin Core хранится более пяти гигабайт.

Однако для Utreexo все равно потребуется, чтобы некоторые узлы хранили все данные UTXO, особенно те узлы, которые обслуживают майнеров и другие операции с быстрой проверкой новых блоков. Это означает, что невыгодные выходы могут стать проблемой для полных узлов даже в том вероятном будущем, когда большинство узлов будут использовать Utreexo.

В правилах политики Bitcoin Core в отношении «пыли» есть одно исключение: скрипты выхода, начинающиеся с `OP_RETURN`, которые называют *выхода-*

ми носителя данных (*data carrier outputs*), могут иметь нулевое значение. Использование OP_RETURN приводит к немедленному сбою скрипта, что бы за ним ни последовало, поэтому эти выходы никогда не могут быть потрачены. Это означает, что полным узлам не нужно следить за ними. Данная возможность используется в Bitcoin Core для хранения пользователями небольших объемов произвольных данных в блокчейне без увеличения размера базы данных UTXO. Поскольку выходы не расходуются, они не являются убыточными – все сатоши, назначенные на них, становятся постоянно нерасходуемыми. По этой причине возможность установить нулевую сумму гарантирует защиту сатоши от уничтожения.

Скрипты выхода

За суммой выхода следует целое число формата compactSize, указывающее на длину *скрипта выхода* (*output script*). Этот скрипт содержит условия, которые необходимо выполнить для использования биткойнов. Согласно правилам консенсуса Биткойна, минимальный размер скрипта выхода равен нулю.

Максимум допустимого размера выходного скрипта по консенсусу зависит от того, когда он проверяется. В явном виде размер выходного скрипта в выходе транзакции не ограничен, но последующая транзакция может потратить только предыдущий выход со скриптом размером 10 000 байт или меньше. В неявном виде выходной скрипт может быть почти таким же большим, как хранящая его транзакция, а транзакция может быть почти такой же большой, как хранящий ее блок.

Выходной скрипт с нулевой длиной может быть потрачен входным скриптом с кодом OP_TRUE. Создать такой входной скрипт может кто угодно, а значит, и потратить пустой выходной скрипт может кто угодно. Существует практически неограниченное количество скриптов, которые может потратить кто угодно, и они известны разработчикам протокола Биткойна как «*потратить может кто угодно*» (*anyone can spends*). Обновления языка скриптов Биткойна часто используют существующий скрипт anyone-can-spend и добавляют к нему новые ограничения, в результате чего он может быть использован только при новых условиях. Разработчики приложений никогда не должны использовать скрипт anyone-can-spend, но если все-таки это произойдет, настоятельно рекомендуем открыто объявить о своих планах пользователям и разработчикам Биткойна, чтобы будущие обновления случайно не вмешались в работу вашей системы.

Политика Bitcoin Core в отношении передачи и майнинга транзакций эффективно ограничивает выходные скрипты лишь несколькими шаблонами, называемыми *стандартными выходными данными транзакций* (*standard transaction outputs*). Это было первоначально реализовано после обнаружения нескольких ранних ошибок в Биткойне, связанных с языком Script, и сохраняется в современном Bitcoin Core для поддержки обновлений anyone-can-spend, а также для поощрения лучших практик размещения условий скриптов в redeem-скриптах P2SH, скриптах свидетелей segwit v0 и скриптах листьев segwit v1 (taproot).

Каждый из современных стандартных шаблонов транзакций будет рассмотрен в главе 7, где также будет рассказано о том, как анализировать скрипты.

Структура свидетеля

Свидетель (witness) в суде – это человек, который подтверждает факт того, что он видел какое-то важное событие. Люди в качестве свидетелей не всегда надежны, поэтому в судах есть различные процедуры допроса, чтобы (в идеале) принимать показания только от тех, кто заслуживает доверия.

Представим, как выглядел бы свидетель решения математической задачи. Например, если бы важной задачей было *x + 2 == 4* и кто-то утверждал, что был свидетелем решения. О чем бы следовало его спросить? Конечно, хотелось бы получить математическое доказательство с указанием значения, которое можно сложить с двумя и получить четыре. Можно даже не прибегать к помощи человека и просто привести в качестве свидетеля предложенное значение *x*. Если бы сказали, что свидетель – это два, то можно было бы заполнить уравнение, проверить его правильность и считать важную проблему решенной.

В случае использования биткойнов важной проблемой, которую нужно решить, является определение наличия разрешения на их использование от человека или людей, которые контролируют эти биткойны. Тысячи полных узлов, которые следят за соблюдением правил консенсуса в Биткойне, не могут опрашивать свидетелей-людей, но зато способны принимать свидетелей, полностью состоящих из данных для решения математических задач. Например, свидетель 2 позволит потратить биткойны, защищенные следующим скриптом:

```
2 OP_ADD 4 OP_EQUAL
```

Разумеется, разрешить расходовать ваши биткойны любому, кто сможет решить простое уравнение, было бы небезопасно. Как будет показано в главе 8, схема не поддающейся подделке цифровой подписи использует уравнение, решить которое сможет только обладатель определенных данных, хранящихся в секрете. Эти секретные данные могут быть использованы с помощью открытого идентификатора. Такой открытый идентификатор называется *открытым ключом* (*public key*), а решение уравнения – *подписью* (*signature*).

Следующий скрипт содержит открытый ключ и код, требующий соответствующей подписи для регистрации данных в расходной транзакции. Как и число 2 в простом примере выше, подпись является нашим свидетелем:

```
<public key> OP_CHECKSIG
```

Свидетели – то есть значения, используемые при решении математических задач для защиты биткойнов, – должны быть включены в используемые ими транзакции для подтверждения полными узлами. В унаследованном формате транзакций, который использовался для всех ранних транзакций Биткойна, подписи и другие данные размещались в поле скрипта входа. Однако с внедрением в Биткойн протоколов контрактов, как было показано в разделе «Изначальная замена транзакций на основе последовательности», разработчики обнаружили несколько существенных проблем с размещением свидетелей в поле скрипта входа.

Циклические зависимости

Многие протоколы контрактов для Биткойна включают в себя серию транзакций, которые подписываются вне очереди. Например, Алиса и Боб хотят внести средства в скрипт, который может быть потрачен только при наличии подписей от них обоих, но при этом каждый из них хочет получить свои деньги обратно, если другой перестанет реагировать. Простое решение – подписывать транзакции вне очереди:

- Tx_0 вносит деньги от Алисы и деньги от Боба в выход со скриптом, требующим для расходования подписи как Алисы, так и Боба;
- Tx_1 проводит предыдущий выход в два выхода, один возвращает Алисе ее деньги, а другой возвращает Бобу его деньги (за вычетом небольшой суммы на комиссию за транзакцию);
- если Алиса и Боб подпишут Tx_1 до подписания Tx_0, то оба гарантированно смогут получить возврат в любое время. Протокол не требует, чтобы кто-то из них доверял другому, поэтому он является *не требующим доверия протоколом* (*trustless protocol*).

Проблема такой структуры со старым форматом транзакций заключается в том, что каждое поле с подписями, включая поле скрипта входа, используется для вычисления идентификатора транзакции (txid). Txid для Tx_0 является частью поля outpoint входа в Tx_1. Это означает, что Алиса и Боб не смогут создать Tx_1 до тех пор, пока не будут известны обе подписи для Tx_0. Но если они будут знать подписи для Tx_0, один из них сможет передать эту транзакцию до подписания транзакции возмещения, что снимет гарантию возмещения. Это и есть *циклическая зависимость* (*circular dependency*).

Изменение транзакций третьей стороной

Иногда более сложная серия транзакций может устранить циклическую зависимость, но тогда многие протоколы столкнутся с новой проблемой: один и тот же скрипт нередко можно выполнить разными способами. Для примера можно рассмотреть простой скрипт из раздела «Структура свидетеля»:

```
2 OP_ADD 4 OP_EQUAL
```

Можно добиться выполнения этого скрипта, указав значение 2 в скрипте входа, но существует несколько способов внести это значение в стек Биткойна. Вот лишь некоторые из них:

```
OP_2
OP_PUSH1 0x02
OP_PUSH2 0x0002
OP_PUSH3 0x000002
...
OP_PUSHDATA1 0x0102
OP_PUSHDATA1 0x020002
...
OP_PUSHDATA2 0x000102
```

```
OP_PUSHDATA2 0x00020002
...
OP_PUSHDATA4 0x0000000102
OP_PUSHDATA4 0x000000020002
...
```

Каждое альтернативное кодирование числа 2 в скрипте входа приводит к созданию немного другой транзакции с совершенно другим txid. Каждая отдельная версия транзакции использует те же входы (поле outpoint), что и все остальные версии транзакций, в результате чего все они *конфликтуют* друг с другом. Только одна версия из множества конфликтующих транзакций может храниться в действующем блокчейне.

Представим, что Алиса создает одну версию транзакции с OP_2 в скрипте входа и с выходом, который оплачивает Боб. Затем Боб немедленно отправляет этот выход Кэрол. Любой человек в сети может заменить `OP_2` на `OP_PUSH1 0x02`, создав конфликт с первоначальной версией Алисы. Если эта конфликтующая транзакция подтверждается, то включить первоначальную версию Алисы в тот же блокчейн не получится, а значит, транзакция Боба не сможет потратить свой выход. Платеж Боба в пользу Кэрол стал недействительным, хотя ни Алиса, ни Боб, ни Кэрол не сделали ничего неправильного. Кто-то, не участвующий в транзакции (третья сторона), смог изменить (мутировать) транзакцию Алисы. Эта проблема называется нежелательным *изменением транзакции третьей стороной* (*third-party transaction malleability*).

Бывают случаи, когда действительно необходимо изменить транзакции, и для этого в Биткойне предусмотрено несколько функций. Прежде всего это хеши подписи (sighash), о которых будет рассказано в разделе «Типы хешей подписи (SIGHASH)». Например, Алиса может использовать sighash, чтобы дать Бобу возможность помочь ей оплатить некоторые транзакции. Это мутирует транзакцию Алисы, но только нужным Алисе образом. По данной причине в этой книге к термину «*изменчивость транзакций*» (*transaction malleability*) иногда добавляется слово «*нежелательная*» (*unwanted*). Даже когда мы и другие технические писатели, публикующие информацию о Биткойне, используем более короткий термин, мы почти наверняка говорим о нежелательном варианте изменчивости.

Изменение транзакций второй стороной

Пока старый формат транзакций оставался единственным, разработчики работали над предложениями по минимизации изменчивости транзакций третьей стороной, такими как BIP62. Однако даже если бы удалось полностью исключить возможность такой изменчивости со стороны третьих лиц, пользователи протоколов контрактов (contract protocols) оказались бы перед другой проблемой: если им требовалась подпись от кого-то другого, задействованного в протоколе, этот человек мог бы сгенерировать альтернативные подписи и изменить txid.

Например, Алиса и Боб внесли свои деньги в скрипт, требующий подписи обоих для их расходования. Они также создали транзакцию возврата, которая позволит каждому из них получить свои деньги обратно в любое время. Алиса решила потратить только часть денег и вместе с Бобом создала цепочку транзакций:

- Tx_0 включает подписи Алисы и Боба и проводит их биткойны на два выхода. Первый выход расходует часть денег Алисы; второй выход возвращает оставшиеся биткойны обратно в скрипт, требующий подписи Алисы и Боба. Перед тем как подписать эту транзакцию, они создают новую транзакцию возврата средств, Tx_1;
- Tx_1 расходует второй выход Tx_0 на два новых выхода: один – Алисе с ее долей совместных средств, другой – Бобу с его долей. Алиса и Боб подписывают эту транзакцию перед подписанием Tx_0.

Здесь нет круговой зависимости, и если отбросить возможность изменчивости транзакций третьей стороной, кажется, что все это должно привести к ситуации с не требующим доверия протоколом (trustless protocol). Однако подписи Биткойна характеризуются тем, что подписывающий должен выбрать большое случайное число при создании своей подписи. Выбор другого случайного числа приведет к созданию другой подписи, даже если все подписываемое останется неизменным. Это сродни рукописной подписи под двумя копиями одного и того же контракта, но при этом каждая из этих физических подписей будет выглядеть немного иначе.

Такая «изменяемость» подписей означает, что если Алиса попытается передать Tx_0 (где содержится подпись Боба), Боб может сгенерировать альтернативную подпись для создания конфликтующей транзакции с другим txid. Если альтернативная версия Tx_0 Боба будет подтверждена, то Алиса не сможет использовать предварительно подписанную версию Tx_1 для получения возврата. Такой тип мутации называется *нежелательной изменчивостью транзакций второй стороны* (*unwanted second-party transaction malleability*).

Сегрегированный свидетель (Segregated Witness)

Еще в 2011 году разработчики протокола придумали решение проблем циклической зависимости, изменяемости от третьей стороны и изменяемости от второй стороны. Идея заключалась в отказе от включения скрипта входа в вычисления, которые приводят к получению txid транзакции. Напомним, что *свидетель* (*witness*) – это абстрактное наименование данных, хранящихся в скрипте входа. Концепция отделения других данных в транзакции от ее свидетеля с целью генерации txid называется *отдельным свидетелем*, или *сегрегированным свидетелем* (*segregated witness*, сокращенно *segwit*).

Наиболее простой способ реализации segwit требует изменения правил консенсуса Биткойна, которые будут несовместимы со старыми полными узлами. Это также называется «хард-форк» (hard fork, букв. «жесткое ответвление»). Такие хард-форки сопряжены со многими трудностями, о которых подробнее рассказывается в разделе «Хард-форки».

Альтернативный метод использования segwit был опубликован в конце 2015 года. Он предполагает изменение правил консенсуса с обратной совместимостью, которое называется «*софт-форк*» (*soft fork*). Обратная совместимость позволяет полным узлам, внедряющим изменение, не принимать никаких блоков, которые полные узлы без изменения сочли бы недействительными. Пока они следуют этому правилу, обновленные полные узлы могут отклонять

блоки, которые старые полные узлы приняли бы. Это дает им возможность ввести новые правила консенсуса (но только если новые полные узлы представляют интересы экономического консенсуса среди пользователей Биткойна). Подробности обновления правил консенсуса Биткойна будут рассмотрены в главе 12.

В основе софт-форк метода использования segwit лежат скрипты выхода типа any-can-spend («потратить может любой, кто захочет»). Скрипт, начинающийся с любого из чисел от 0 до 16 и содержащий от 2 до 40 байт данных, определяется как шаблон выходного скрипта segwit (segwit output script template). Число указывает на его версию (например, 0 – это segwit версии 0, или *segwit v0*). Эти данные называются «*программой свидетеля*» (*witness program*). Можно также упаковать шаблон segwit в обязательство P2SH, но в данной главе этот вопрос не рассматривается.

Старые узлы могут использовать эти шаблоны скриптов выхода с пустым скриптом входа. Новые узлы, ознакомленные с новыми правилами segwit, обязаны расходовать любые платежи по шаблону скрипта выхода segwit только с пустым скриптом входа. Обратите внимание на разницу: старые узлы *разрешают* пустой скрипт входа; новые узлы *требуют* пустой скрипт входа.

Пустой скрипт входа не позволяет свидетелям оказывать влияние на txid, устраняя циклические зависимости, изменяемость транзакций третьей стороной и изменяемость транзакций второй стороной. Однако из-за отсутствия возможности помещать данные в скрипт входа пользователям шаблонов скриптов выхода segwit требуется новое поле. Это поле называется *структурой свидетеля* (*witness structure*).

Введение программ-свидетелей и структуры свидетелей усложняет Биткойн, но при этом соответствует сложившейся тенденции к увеличению абстракции. В главе 4, например, говорилось о том, что в первоначальном документе о Биткойне описывалась система, в которой биткойны принимались на открытые ключи (pubkeys) и расходовались с помощью подписей (sigs). Открытый ключ указывал, кто имеет право тратить биткойны (тот, кто контролировал соответствующий секретный ключ), а подпись удостоверяла, что транзакция была произведена тем, кто контролировал секретный ключ. Для повышения гибкости этой системы в первоначальном релизе Биткойна были введены скрипты, позволяющие получать биткойны на скрипты выхода и тратить их на скрипты входа. Позже опыт использования протоколов контрактов привел к появлению возможности получать биткойны в программы свидетелей и тратить их с помощью структуры свидетелей. Термины и поля, используемые в разных версиях Биткойна, приведены в табл. 6.2.

Таблица 6.2. Термины для авторизации и аутентификации данных в разных версиях Биткойна

	Авторизация	**Аутентификация**
Документация	Открытый ключ	Подпись
Исходный (устаревший)	Скрипт выхода	Скрипт входа
Segwit	Программа-свидетель	Структура свидетеля

Сериализация структуры свидетеля

По аналогии с полями входов и выходов, структура свидетеля содержит несколько полей, поэтому для начала рассмотрим карту этих байтов из транзакции Алисы на рис. 6.5.

Рис. 6.5 ❖ Карта байтов структуры свидетеля из транзакции Алисы

В отличие от полей входов и выходов, общая структура свидетелей не начинается с указания общего количества содержащихся в ней стеков свидетелей. Вместо этого она определяется полем «входы» (inputs): на каждый вход транзакции приходится один стек свидетелей.

Структура свидетелей для конкретного входа начинается с подсчета количества содержащихся в нем элементов. Они называются *элементами свидетеля* (*witness items*). Подробно о них будет рассказано в главе 7, а пока достаточно знать, что каждому элементу свидетеля предшествует целое число compactSize, указывающее на его размер.

Устаревшие входы не содержат никаких элементов свидетелей, поэтому их стек свидетелей целиком состоит из нуля (0x00).

Транзакция Алисы содержит один вход и один элемент свидетеля.

Время блокировки

Последним полем в сериализованной транзакции является ее время блокировки. Это поле входило в исходный формат сериализации Биткойна, но поначалу оно соблюдалось только в политике Биткойна по отбору транзакций для майнинга. Самый ранний из известных софт-форков биткойна добавил правило, согласно которому, начиная с блока высотой 31 000, запрещалось включать транзакцию в блок, если она не удовлетворяет одному из следующих правил:

- транзакция указывает, что она может быть включена в любой блок, устанавливая время блокировки на 0;
- транзакция указывает, что должна ограничить блоки, в которые она может быть включена, устанавливая время блокировки менее 500 000 000. В этом случае транзакция может быть включена только в блок, высота

которого равна времени блокировки или больше. Например, транзакция со временем блокировки 123 456 может быть включена в блок 123 456 или любой более поздний блок;

- транзакция указывает, что намерена ограничить время своего включения в блокчейн, установив время блокировки в значение 500 000 000 или больше. В этом случае поле обрабатывается как время эпохи (количество секунд начиная с даты 01-01-1970 00:00 по Гринвичу), и транзакция может быть включена только в блок с *медианным прошедшим временем* (*Median Time Past*, *MTP*), превышающим время блокировки. Правила для MTP описаны в разделе «Медианное время прошедшего периода (MTP)».

Транзакции coinbase

Первая транзакция в каждом блоке – это особый случай. В большей части старой документации она называется *транзакцией поколения* (*generation transaction*), а в большей части новой документации – *транзакцией coinbase* (*coinbase transaction*, не путать с транзакциями, созданными компанией под названием «Coinbase»).

Транзакции coinbase создаются майнером блока, в который они включены, и дают майнеру возможность претендовать на любые комиссии, выплачиваемые транзакциями в этом блоке. Кроме того, до блока 6 720 000 майнерам разрешается требовать вознаграждение из биткойнов, которые никогда ранее не находились в обращении, называемое *субсидией блока* (*block subsidy*). Общая сумма, которую майнер может потребовать за блок, – сочетание платы и субсидии – называется *вознаграждением за блок* (*block reward*).

К числу особых правил для транзакций coinbase относятся:

- они могут иметь только один вход;
- единственный вход должен иметь поле outpoint с нулевым txid (состоящим исключительно из нулей) и максимальным выходным индексом (0xffffffff). Это не позволит транзакции coinbase ссылаться на выход предыдущей транзакции, что (как минимум) вызовет путаницу, учитывая, что транзакция coinbase выплачивает комиссии и субсидии;
- поле, которое в обычной транзакции содержало бы скрипт входа, называется *coinbase*. Именно это поле и дает название транзакции coinbase. Поле coinbase должно содержать не менее 2 байт и не более 100 байт. Этот скрипт не выполняется, но к нему применяются ограничения на количество операций проверки подписи (sigops), установленные в традиционных транзакциях. По этой причине любые произвольные данные, помещенные в него, должны сопровождаться операционным кодом data-pushing. Поскольку софт-форк 2013 года определен в BIP34, первые несколько байтов этого поля должны соответствовать дополнительным правилам, которые будут описаны в разделе «Данные coinbase»;
- сумма выходов не должна превышать величину комиссии, собранной со всех транзакций в этом блоке, плюс субсидия. Субсидия начинается

с 50 BTC за блок и уменьшается вдвое через каждые 210 000 блоков (примерно раз в четыре года). Значения субсидий округляются до ближайшего сатоши;

- начиная с софт-форка segwit 2017 года, описанного в BIP141, любой блок с транзакцией, тратящей выход segwit, должен содержать выход на транзакцию coinbase, которая фиксирует все транзакции в блоке (включая их свидетелей). Это обязательство будет рассмотрено в главе 12.

Транзакция coinbase может иметь любые другие выходы, которые были бы действительны в обычной транзакции. Однако транзакция с одним из этих выходов не может быть включена ни в один блок до тех пор, пока транзакция coinbase не получит 100 подтверждений. Это называется *правилом зрелости* (*maturity rule*), а выходы транзакций coinbase, которые еще не получили 100 подтверждений, называются *незрелыми* (*immature*).

Большинству программ для работы с биткойном не нужно иметь дело с транзакциями coinbase, но их особый характер означает, что иногда они могут быть причиной нестандартных проблем в программах, которые не рассчитаны на работу с ними.

Объем данных транзакции: weight и vbyte

Каждый блок Биткойна ограничен по объему данных о проводимых транзакциях, поэтому большинство программ для Биткойна должны иметь возможность измерять созданные или обработанные транзакции. Современная единица измерения для Биткойна называется *вес* (*weight*). Альтернативный вариант веса – «*вбайт*» (*vbyte*), где четыре единицы веса равны одному vbyte. Это позволяет легко сравнить его с оригинальной единицей измерения в байтах, используемой в старых блоках Биткойна.

Вес блоков ограничен 4 млн. Заголовок блока имеет вес 240. Дополнительное поле, счетчик транзакций, имеет вес 4 или 12 единиц. Весь оставшийся вес может быть использован для данных транзакции.

Для расчета веса конкретного поля транзакции размер этого сериализованного поля в байтах умножается на некоторый коэффициент. Чтобы вычислить вес транзакции, нужно просуммировать значения веса всех ее полей. Коэффициенты для каждого из полей транзакции приведены в табл. 6.3. Для сравнения также рассчитан вес каждого поля в примере транзакции от Алисы к Бобу, приведенном в этой главе.

Коэффициенты и соответствующие им поля были подобраны таким образом, чтобы уменьшить вес, используемый при расходовании UTXO. Это помогает предотвратить создание невыгодных выходов, описанных в разделе «Невыгодные выходы и запрещенная пыль».

Можно проверить вычисления веса, получив итоговую сумму транзакции Алисы из Bitcoin Core:

```
$ bitcoin-cli getrawtransaction 466200308696215bbc949d5141a49a41\
38ecdfdfaa2a8029c1f9bcecd1f96177 2 | jq .weight
569
```

Таблица 6.3. Весовые коэффициенты для всех полей транзакции Биткойна

Поле	Коэффициент	Вес в Tx Алисы
Версия	4	16
Маркер и флаг	1	2
Количество входов	4	4
Outpoint	4	144
Скрипт входа	4	4
Sequence	4	16
Количество выходов	4	4
Сумма	4	64 (2 выхода)
Скрипт выхода	4	232 (2 выхода с разными скриптами)
Счетчик свидетелей	1	1
Количество свидетелей	1	66
Время блокировки	4	16
Всего	н/д	569

Транзакция Алисы из примера 6.1 в начале этой главы представлена в весовых единицах на рис. 6.6. Можно увидеть, как работает этот коэффициент, сравнив разницу в размерах различных полей на этих двух изображениях.

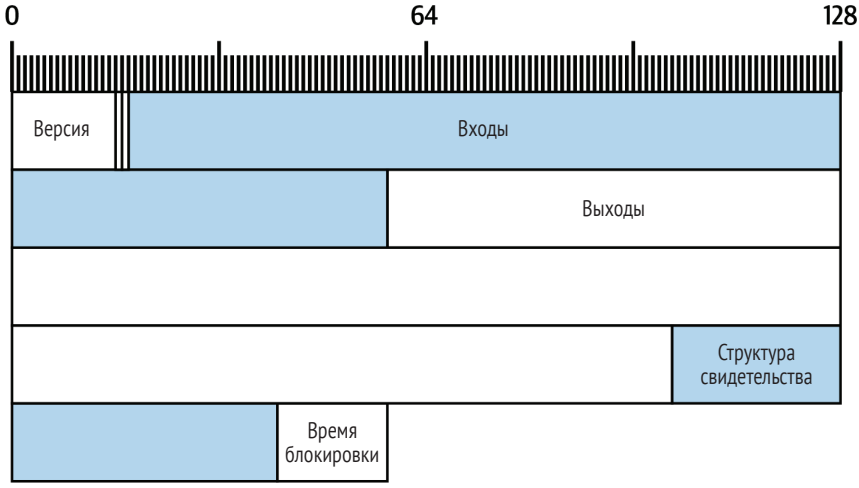


Рис. 6.6 ❖ Карта байтов транзакции Алисы

Унаследованная сериализация

Формат сериализации, описанный в этой главе, используется для большинства новых транзакций Биткойна на момент написания данной книги, но более старый формат сериализации все еще применяется для многих транзакций. Этот

старый формат, называемый *унаследованной (устаревшей) сериализацией* (*legacy serialization*), должен использоваться в P2P-сети Bitcoin для любой транзакции с пустой структурой свидетелей (которая действительна только в том случае, если транзакция не проводит никаких программ-свидетелей).

Унаследованная сериализация не включает поля маркера, флага и структуры свидетеля.

В этой главе на примере транзакции были рассмотрены все эти поля и было показано, как они передают полным узлам подробную информацию о передаваемых между пользователями биткойнах. Однако здесь лишь вкратце были рассмотрены скрипты выхода, скрипты входа и структура свидетелей. Все они позволяют задавать условия, ограничивающие возможности пользователей по расходованию биткойнов, и удовлетворять их. Понимание принципов построения и использования этих условий необходимо для гарантии того, что только Алиса может тратить свои биткойны. Поэтому они станут предметом изучения в следующей главе.

Глава 7

Авторизация и аутентификация

После получения биткойнов нужно решить, кто будет иметь право их тратить. Это и есть авторизация (authorization). Также необходимо определить способ, с помощью которого полные узлы будут отличать авторизованных плательщиков от всех остальных. Это и есть аутентификация (authentication). Указания по авторизации и доказательства аутентификации тратящего лица будут проверяться тысячами независимых полных узлов. И все они должны прийти к одному и тому же выводу о наличии авторизации и аутентификации у потратившего средства, чтобы содержащая его транзакция считалась действительной.

В первоначальном описании Биткойна в качестве средства авторизации использовался открытый ключ. Алиса платила Бобу, помещая его открытый ключ в выход транзакции. Подтверждение подлинности исходило от Боба в виде подписи, удостоверяющей транзакцию, например, от Боба к Кэрол.

В первоначальной версии Биткойна был реализован более гибкий механизм авторизации и аутентификации. Улучшения, внесенные с тех пор, только увеличили эту гибкость. В этой главе приведены функции, наиболее часто используемые для этих целей, и показано, как они применяются.

Скрипты транзакций и язык скриптов

В первоначальной версии Биткойна был представлен новый язык программирования под названием *Script* – язык на основе стека, похожий на Forth. На этом скриптовом языке пишут и скрипты выхода, и унаследованные скрипты входа, используемые в транзакциях.

Script – очень простой язык. Он предполагает минимальную обработку данных и не может выполнять многие из тех причудливых вещей, которые могут делать современные языки программирования.

В то время когда устаревшие нынче транзакции (legacy transactions) были наиболее распространенным типом операций, большинство из них при обработке в сети Биткойн имели форму «оплата на биткойн-адрес Боба» и исполь-

зовали скрипт оплаты на хеш открытого ключа (pay to public key hash, P2PKH). Однако транзакции Биткойна не ограничиваются лишь скриптом «оплаты на биткойн-адрес Боба». На самом деле скрипты могут быть созданы для описания огромного количества сложных условий. Чтобы понять эти более сложные скрипты, нужно прежде всего разобраться в основах скриптов транзакций и языка Script.

В этом разделе будут рассмотрены основные компоненты языка сценариев транзакций Биткойна, а также будет показано, как с его помощью можно задать условия для расходования средств и как эти условия могут быть выполнены.

Проверка транзакций Биткойна не строится на статичном шаблоне, а осуществляется с помощью скриптового языка. Этот язык позволяет выразить практически бесконечное множество условий.

Неполнота по Тьюрингу

Язык скриптов транзакций Биткойна содержит множество операторов, но при этом намеренно ограничен в одном важном аспекте – в нем нет циклов или сложных возможностей управления процессами, кроме управления потоком по условию. Это обеспечивает языку *неполноту по Тьюрингу* (*not Turing Complete*), то есть ограниченную сложность скриптов и предсказуемое время их выполнения. Язык Script не является языком общего назначения. Благодаря этим ограничениям этот язык не может быть использован для создания бесконечного цикла или другой формы «логической бомбы», которую можно было бы внедрить в транзакцию и тем самым вызвать атаку типа «отказ в обслуживании» (DoS) на сеть Биткойн. Следует помнить, что каждая транзакция проверяется каждым полным узлом сети Биткойн. Ограниченный язык не позволяет использовать механизм проверки транзакций в качестве уязвимости.

Верификация без сохранения состояния

Язык скриптов транзакций Биткойна не имеет состояния до выполнения скрипта и не сохраняет состояние после выполнения скрипта. Вся необходимая для выполнения скрипта информация содержится в самом скрипте и выполняющей его транзакции. Скрипт будет предсказуемо выполняться одинаковым образом на любой системе. Если система проверила скрипт, можно быть уверенным, что любая другая система в сети Биткойн также его верифицирует, а это значит, что корректная транзакция действительна для всех, и все об этом знают. Такая предсказуемость результатов является существенным преимуществом системы Биткойн.

Структура скриптов

Традиционная система проверки транзакций Биткойна опирается на два вида скриптов для подтверждения транзакций: скрипт выхода (output script) и скрипт входа (input script).

Скрипт выхода определяет условия, которые должны быть выполнены для расходования средств в будущем, например кому разрешено расходовать средства и как происходит их аутентификация.

Скрипт входа – это скрипт, который соответствует условиям, заданным в скрипте выхода, и разрешает расходование средств. Скрипты входа являются частью каждого входа транзакции. Чаще всего в унаследованных транзакциях они содержат цифровую подпись, созданную кошельком пользователя на основе его секретного ключа, но не все скрипты входа должны содержать подписи.

Каждый проверяющий узел Биткойна подтверждает правильность транзакций, выполняя скрипты выхода и входа. Как уже говорилось в главе 6, каждый вход содержит поле outpoint, которое ссылается на выход предыдущей транзакции. Вход также содержит скрипт входа. Программа валидации копирует скрипт входа, извлекает UTXO, на который ссылается вход, и копирует скрипт выхода из этого UTXO. Затем скрипты входа и выхода выполняются вместе. Вход считается корректным, если скрипт входа удовлетворяет условиям скрипта выхода (см. раздел «Раздельное выполнение скриптов выхода и входа»). Все входы проверяются независимо друг от друга в рамках полной валидации транзакции.

Следует отметить, что при выполнении всех предшествующих шагов создаются копии всех данных. Исходные данные в предыдущем выходе и актуальном входе никогда не изменяются. В частности, предыдущий выход неизменен и не подвержен влиянию неудачных попыток его использования. Только корректная транзакция, отвечающая условиям скрипта выхода, приводит к признанию выхода «потраченным».

На рис. 7.1 приведен пример скриптов выхода и входа для наиболее распространенного типа унаследованной транзакции Биткойна (платеж в хеш-сумму с открытым ключом), а также показан комбинированный скрипт, полученный в результате объединения скриптов перед их валидацией.

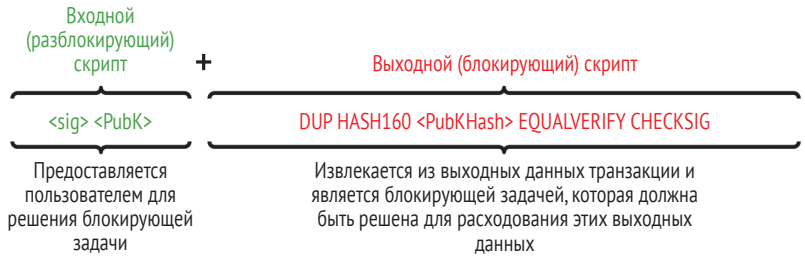


Рис. 7.1 ❖ Объединение скриптов входа и выхода для оценки скрипта транзакции

Стек выполнения скрипта

Язык скриптов Биткойна основан на использовании структуры данных вида «*стек*» (*stack*), потому его и называют языком программирования на основе стека. Стек – это очень простая структура данных, которую можно представить в виде колоды карт. Стек допускает две стандартные операции: push и pop. Операция push добавляет элемент на вершину стека. Операция pop удаляет верхний элемент из стека.

Скриптовый язык обрабатывает скрипт, обращаясь к каждому элементу слева направо. Числа (константы данных) помещаются в стек. Операторы «заталкивают» (push) или «выталкивают» (pop) один или несколько параметров из стека, выполняют над ними действия и в конечном счете «заталкивают» результат в стек. Например, оператор `OP_ADD` берет из стека два элемента, складывает их и помещает полученную сумму в стек.

Условные операторы выполняют проверку условия, получая логический результат `TRUE` (ИСТИНА) или `FALSE` (ЛОЖЬ). Например, `OP_EQUAL` выводит из стека два элемента и вставляет `TRUE` (`TRUE` обозначается цифрой 1), если они равны, или `FALSE` (обозначается 0), если они не равны. Скрипты биткойн-транзакций обычно содержат условный оператор для получения результата `TRUE` в случае корректной транзакции.

Простой скрипт

Теперь можно рассмотреть полученные знания о скриптах и стеках в простых примерах.

Как показано на рис. 7.2, скрипт `2 3 OP_ADD 5 OP_EQUAL` наглядно демонстрирует работу оператора арифметического сложения `OP_ADD`, который складывает два числа и помещает результат в стек, а затем условного оператора `OP_EQUAL`, который выполняет сравнение полученной суммы с числом 5. Для краткости в примерах этой книги префикс `OP_` может иногда отсутствовать. Более подробную информацию о доступных операторах и функциях скриптов можно найти на странице скриптов в Bitcoin Wiki (https://en.bitcoin.it/wiki/Script).

Большинство устаревших скриптов выхода ссылаются на хеш открытого ключа (по сути, устаревший биткойн-адрес), тем самым требуя подтверждения права собственности для расходования средств. Однако скрипт не обязательно должен быть таким сложным. Допустима любая комбинация скриптов выхода и входа, которая приводит к значению TRUE. Простая арифметика из примера языка скриптов также является допустимым скриптом.

В качестве скрипта выхода можно использовать часть скрипта из арифметического примера:

```
3 OP_ADD 5 OP_EQUAL
```

который может быть обеспечен транзакцией, содержащей вход со сценарием входа:

```
2
```

Программа валидации объединяет скрипты:

```
2 3 OP_ADD 5 OP_EQUAL
```

Как показано на рис. 7.2, результатом выполнения этого скрипта является OP_TRUE, что подтверждает корректность транзакции. Хотя это допустимый скрипт выхода транзакции, следует заметить, что полученный UTXO может быть потрачен кем угодно, чьих познаний в арифметике хватает для определения соответствия числа 2 условиям скрипта.

Рис. 7.2 ❖ Скрипт проверки Биткойна, выполняющий простые вычисления

Транзакции считаются корректными, если результат наверху стека имеет значение `TRUE`, то есть любое ненулевое значение. Транзакции считаются недействительными, если верхнее значение в стеке равно `FALSE` (нулевое значение, или пустой стек), выполнение скрипта было явно остановлено оператором (например, `VERIFY`, `OP_RETURN`) или скрипт не был семантически корректен (например, содержал оператор `OP_IF`, который не завершился оператором `OP_ENDIF`). Подробности см. на странице скриптов в Bitcoin Wiki (https://en.bitcoin.it/wiki/Script).

Ниже приведен более сложный скрипт, вычисляющий 2 + 7 – 3 + 1. Следует заметить, что когда скрипт содержит несколько операторов подряд, стек позволяет использовать результаты одного оператора в следующем:

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Попробуйте проверить предыдущий скрипт самостоятельно, с помощью карандаша и бумаги. По окончании выполнения скрипта в стеке должно остаться значение TRUE.

Раздельное выполнение скриптов выхода и входа

В первоначальной версии клиента Биткойна скрипты выхода и входа были объединены и выполнялись последовательно. По соображениям безопасности в 2010 году это было изменено. Причиной тому стала уязвимость, известная как ошибка `1 OP_RETURN`. В нынешней реализации скрипты выполняются по отдельности с перемещением стека между двумя исполнениями.

Сначала с помощью исполнительного механизма стека выполняется скрипт входа. Если он выполнен без ошибок и у него не осталось операций, выполняется копирование стека и переход к выполнению скрипта выхода. Если результатом его работы с данными стека, скопированными из скрипта входа, является значение `TRUE`, то это означает, что скрипт входа успешно справился с условиями, наложенными скриптом выхода, и, следовательно, вход является валидной авторизацией для проведения UTXO. Если после выполнения комбинированного скрипта результат отличается от `TRUE`, значит, скрипт входа признан недействительным, поскольку он не смог соответствовать условиям расходования, заданным на выходе.

Скрипт Pay to Public Key Hash (P2PKH)

Скрипт pay to public key hash (P2PKH) использует скрипт выхода, содержащий хеш с обязательствами открытого ключа. P2PKH наиболее известен как основа устаревшего (legacy) биткойн-адреса. Выход P2PKH можно потратить с предъявлением открытого ключа, который соответствует обязательству хеша, а также цифровой подписи, созданной соответствующим секретным ключом (см. главу 8). Рассмотрим пример скрипта выхода P2PKH:

```
OP_DUP OP_HASH160 <Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Хеш ключа – это данные, которые будут закодированы в legacy-адресе формата base58check. Большая часть приложений отобразит *хеш открытого ключа* (*public key hash*) в скрипте с шестнадцатеричной кодировкой, а не в привычном для биткойн-адресов формате base58check, который начинается с «1».

Условия этого предшествующего скрипта выхода могут быть выполнены с помощью скрипта входа вида:

```
<Signature> <Public Key>
```

Оба скрипта вместе образуют следующий комбинированный скрипт подтверждения:

```
<Sig> <Pubkey> OP_DUP OP_HASH160 <Hash> OP_EQUALVERIFY OP_CHECKSIG
```

В результате будет получено значение TRUE, если в скрипте есть действительная подпись от секретного ключа Боба, соответствующая хешу открытого ключа, установленному в качестве ограничивающего условия.

На рис. 7.3 и 7.4 показано (в двух частях) пошаговое выполнение комбинированного скрипта, который подтверждает корректность транзакции.

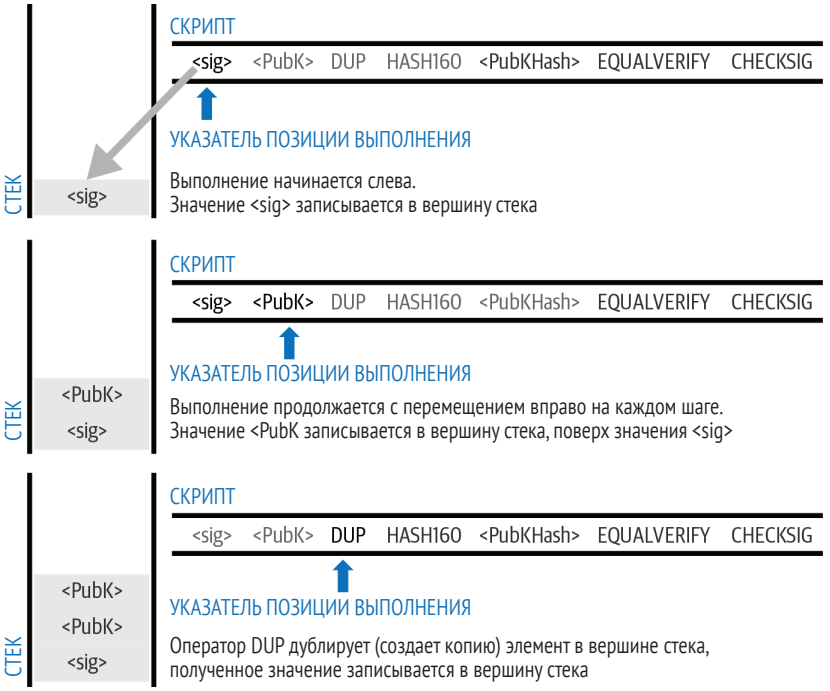


Рис. 7.3 ❖ Проверка скрипта для транзакции P2PKH (часть 1 из двух)

Скриптовые мультиподписи

Скрипты мультиподписи (скрипты с несколькими подписями – multisignature scripts) задают условие, по которому в скрипт записывается *k* открытых ключей, и при этом не менее *t* из них должны предоставить подписи для расходования средств – это называют параметром «*t* из *k*» (*t*-of-*k*). Например, мультиподпись 2-of-3 – это скрипт, в котором три открытых ключа указаны в качестве потенциальных подписывающих лиц и по крайней мере два из них должны быть задействованы для создания подписей под допустимой транзакцией для расходования средств.

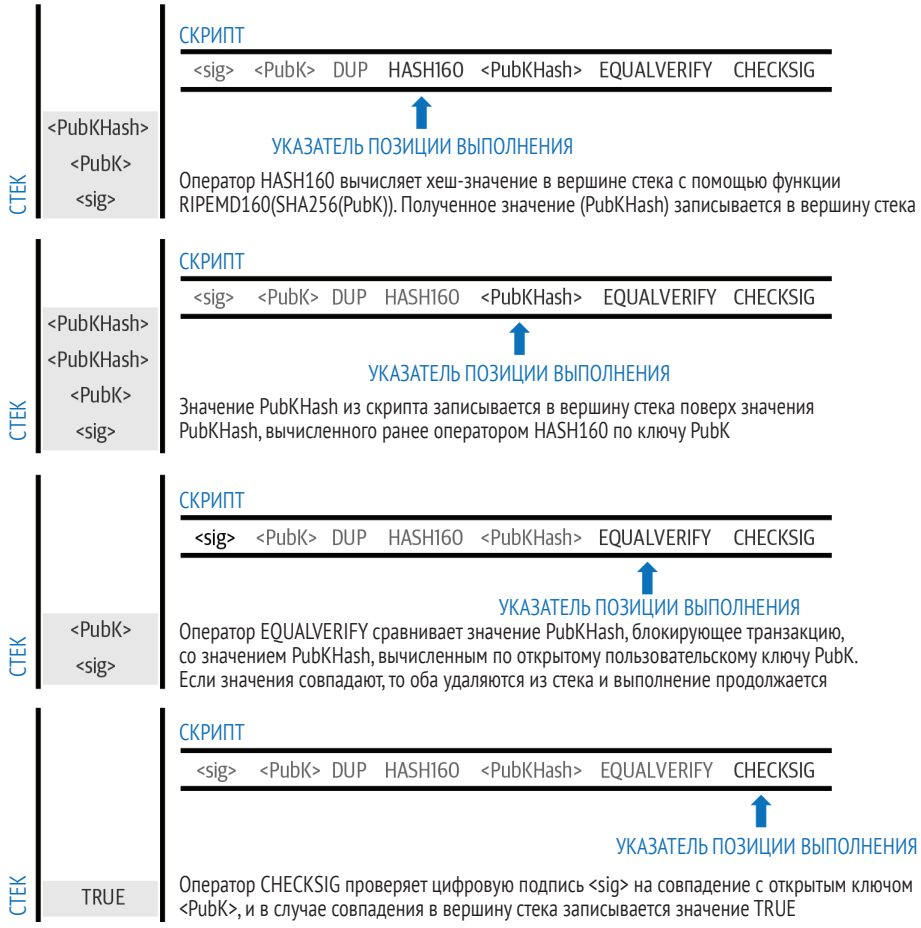


Рис. 7.4 ❖ Проверка скрипта для транзакции P2PKH (часть 2 из двух)

В документации по Биткойну, как и в предыдущих изданиях этой книги, для обозначения традиционной мультиподписи применяется термин «m-of-n». Однако при произнесении сложно различить «m» и «n», поэтому в данном случае используется альтернатива «*t*-of-*k*». Обе фразы относятся к одному и тому же типу схемы подписи.

В общем виде скрипт вывода, задающий условие мультиподписи *t*-of-*k*, выглядит так:

```
t <Public Key 1> <Public Key 2> ... <Public Key k> k OP_CHECKMULTISIG
```

где *k* – общее количество указанных открытых ключей, а *t* – минимальное количество подписей, необходимое для выполнения выхода.

Скрипт выхода, задающий условие мультиподписи «2 из 3», выглядит так:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 OP_CHECKMULTISIG
```

Приведенный выше скрипт выхода можно обеспечить с помощью скрипта входа, содержащего подписи:

```
<Signature B> <Signature C>
```

или любую комбинацию из двух подписей секретных ключей, соответствующих трем перечисленным открытым ключам.

Эти два скрипта вместе образуют комбинированный скрипт проверки:

```
<Sig B> <Sig C> 2 <Pubkey A> <Pubkey B> <Pubkey C> 3 OP_CHECKMULTISIG
```

При выполнении этот комбинированный скрипт будет выдавать значение TRUE, если в скрипте входа есть две корректные подписи секретных ключей, соответствующих двум из трех открытых ключей, установленных в качестве условия блокировки.

В настоящее время политика Bitcoin Core в отношении пересылки транзакций ограничивает скрипты вывода мультиподписей максимум тремя выбранными открытыми ключами. Это означает, что можно сделать любую мультиподпись – от «1 из 1» до «3 из 3» – или любую комбинацию в этом диапазоне. Чтобы узнать, что в настоящее время принимается сетью, можно воспользоваться функцией IsStandard(). Обратите внимание, что ограничение в три ключа применяется только к стандартным (также известным как «чистым» – bare) скриптам с мультиподписью, а не к скриптам, заключенным в другую структуру, например P2SH, P2WSH или P2TR. Скрипты с мультиподписью P2SH ограничены как политикой, так и консенсусом до 15 ключей, что позволяет использовать мультиподпись до количества «15 из 15». Подробнее о P2SH рассказано в разделе «Скрипт Pay to Script Hash (P2SH)». Все остальные скрипты ограничены консенсусом до 20 ключей на каждую операцию OP_CHECKMULTISIG или OP_CHECKMULTISIGVERIFY, однако один скрипт может включать несколько таких операций.

Нестандартное выполнение CHECKMULTISIG

В работе OP_CHECKMULTISIG есть одна особенность, которая требует некоторого вмешательства. При выполнении OP_CHECKMULTISIG ему нужно в качестве параметров $t + k + 2$ элементов из стека. Однако из-за этой особенности OP_CHECKMULTISIG выдает одно дополнительное значение или на одно значение больше, чем ожидалось.

Рассмотрим это подробнее на примере предыдущей процедуры проверки:

```
<Sig B> <Sig C> 2 <Pubkey A> <Pubkey B> <Pubkey C> 3 OP_CHECKMULTISIG
```

Сначала OP_CHECKMULTISIG выбирает верхний элемент, который равен k (в этом примере «3»). Затем он открывает k элементов, которые представляют собой открытые ключи с правом подписи; в этом примере это открытые ключи A, B и C. Потом он открывает один элемент, который равен t и обозначает кворум (сколько подписей требуется). В этом примере $t = 2$. В данный момент OP_CHECKMULTISIG должен открыть последние t элементов, которые являются подписями, и проверить, являются ли они корректными. К сожалению, странная особенность в его реализации приводит к тому, что OP_CHECKMULTISIG выводит на один элемент больше (всего $t + 1$), чем следует. Лишний элемент называется *пустым элементом стека* (*dummy stack element*). Он игнорируется при проверке подпи-

сей, поэтому не оказывает прямого влияния на работу OP_CHECKMULTISIG. Однако присутствие этого пустого элемента необходимо: если его не будет в момент проверки OP_CHECKMULTISIG на пустом стеке, это приведет к ошибке в работе стека и сбою скрипта (пометка транзакции как некорректной). Поскольку пустой элемент игнорируется, он может быть любым. На ранних этапах было принято использовать OP_0, который позже стал правилом политики пересылки и, в конце концов, правилом консенсуса (с внедрением BIP147).

Так как извлечение пустого элемента является частью правил консенсуса, воспроизводить его придется всегда. Поэтому скрипт должен выглядеть так:

```
OP_0 <Sig B> <Sig C> 2 <Pubkey A> <Pubkey B> <Pubkey C> 3 OP_CHECKMULTISIG
```

Таким образом, скрипт входа, фактически используемый для мультиподписи, выглядит не так:

```
<Signature B> <Signature C>
```

а иначе:

```
OP_0 <Sig B> <Sig C>
```

Кто-то полагает, что эта особенность является ошибкой в исходном коде Биткойна, но существует и другое правдоподобное объяснение. Проверка *t*-of-*k* подписей может потребовать гораздо больше, чем просто *t* или *k* операций проверки подписи. Рассмотрим простой пример «1 из 5» со следующим комбинированным скриптом:

```
<dummy> <Sig4> 1 <key0> <key1> <key2> <key3> <key4> 5 OP_CHECKMULTISIG
```

Сначала подпись проверяется по нулевому ключу – key0, затем по key1 и далее по остальным, после чего сравнение производится с соответствующим key4. Это означает, что необходимо выполнить пять операций проверки подписи – и это притом, что подпись всего одна. Одним из способов устранить эту избыточность было бы предоставление OP_CHECKMULTISIG разметки с указанием соответствия предоставленной подписи открытому ключу. Это позволило бы OP_CHECKMULTISIG провести всего лишь три операции проверки подписи. Не исключено, что изначально создатель Биткойна добавил лишний элемент (который мы теперь называем пустым элементом стека) в первоначальную версию Биткойна, чтобы впоследствии дополнить его функцией передачи разметки в более позднем софт-форке. Однако эта функция так и не была реализована, а обновление правил консенсуса BIP147 в 2017 году сделало невозможным добавление данной функции в будущем.

Только автор концепции Биткойна мог бы объяснить нам, был ли этот пустой элемент стека результатом ошибки или планом на будущее обновление. В нашей книге это просто называется особенностью.

С этого момента, если вам встретится скрипт мультиподписи (multisig), стоит ожидать появления дополнительного OP_0 в самом его начале. Его единственное назначение – обходной путь для устранения странности в правилах консенсуса.

Скрипт Pay to Script Hash (P2SH)

Скрипт pay to script hash (P2SH) был представлен в 2012 году в качестве нового эффективного типа операций, существенно упрощающего работу со сложными скриптами. Для пояснения необходимости применения P2SH обратимся к практическому примеру.

Мохаммед – импортер электроники из Дубая. Для своих корпоративных счетов компания Мохаммеда широко использует функцию мультиподписи Биткойна. Скрипты с мультиподписью относятся к наиболее распространенным способам применения расширенных скриптовых возможностей Биткойна и обладают огромным потенциалом. Компания Мохаммеда использует скрипт с мультиподписью для всех клиентских платежей. Каждый клиентский платеж блокируется так, чтобы для его проведения требовалось не менее двух подписей. Мохаммед, три его партнера и их юрист могут поставить по одной подписи. Подобная схема с несколькими подписями (мультиподписью – multisignature) обеспечивает корпоративный контроль и предохраняет от краж, растрат или потерь.

Получившийся скрипт довольно длинный и выглядит так:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key>
<Partner3 Public Key> <Attorney Public Key> 5 OP_CHECKMULTISIG
```

Несмотря на широкие возможности скриптов с мультиподписью, на практике они довольно громоздки. С учетом предыдущего скрипта Мохаммеду пришлось бы передать его каждому клиенту перед оплатой. Всем клиентам пришлось бы использовать специальные программы для биткойн-кошельков с возможностью создания пользовательских скриптов транзакций. Кроме того, результирующая транзакция была бы примерно в пять раз больше, чем простая платежная операция, поскольку этот скрипт содержит очень длинные открытые ключи. Бремя обработки этих дополнительных данных ляжет на плечи клиента в виде дополнительной комиссии за транзакцию. Наконец, такой скрипт большой транзакции будет храниться в комплекте UTXO на каждом полном узле, пока не будет истрачен. Все эти факторы затрудняют использование сложных скриптов выхода на практике.

Для решения этих практических задач и для упрощения использования сложных скриптов в качестве платежа на биткойн-адрес с одним ключом был разработан P2SH. При платежах P2SH сложный скрипт заменяется обязательством (commitment) – дайджестом криптографического хеша (digest of a cryptographic hash). При последующем проведении транзакции, которая должна потратить UTXO, в ней должен содержаться соответствующий обязательству скрипт, а также соответствующие ему данные. Иными словами, P2SH подразумевает «оплату на скрипт, соответствующий этому хешу, – скрипт, который будет представлен позже, когда этот выход будет потрачен».

В транзакциях P2SH скрипт, заменяемый хешем, называется *скриптом погашения* (*redeem script*), поскольку он представляется системе в процессе погашения, а не как скрипт выхода. В табл. 7.1 показан скрипт без P2SH, а в табл. 7.2 – тот же скрипт, закодированный с P2SH.

Таблица 7.1. Сложный скрипт без P2SH

Скрипт выхода	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG.
Скрипт входа	Sig1 Sig2

Таблица 7.2. Сложный скрипт в виде P2SH

Скрипт погашения	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG
Скрипт выхода	OP_HASH160 <20-byte hash of redeem script> OP_EQUAL
Скрипт входа	Sig1 Sig2 <redeem script>

Как видно из таблиц, при использовании P2SH комплексный скрипт с подробным описанием условий расходования средств (redeem script) не представлен в скрипте выхода. Вместо этого в скрипте выхода есть только его хеш, а сам скрипт погашения появляется позже в качестве части скрипта входа, в момент расходования выхода. В результате бремя комиссий и сложностей перекладывается с отправителя средств на получателя транзакции.

Теперь рассмотрим компанию Мохаммеда, комплексный скрипт с мультиподписью и результирующие скрипты P2SH.

Во-первых, скрипт с мультиподписью, который компания Мохаммеда использует для всех входящих платежей от клиентов:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key>
<Partner3 Public Key> <Attorney Public Key> 5 OP_CHECKMULTISIG
```

Весь этот скрипт можно представить в виде 20-байтового криптографического хеша, если сначала применить алгоритм хеширования SHA256, а затем к результату применить алгоритм RIPEMD-160. К примеру, начиная с хеша скрипта погашения Мохаммеда:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

Транзакция P2SH выполняет блокировку выхода на этот хеш, заменяя более длинный скрипт погашения с помощью специального шаблона скрипта выхода:

```
OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL
```

который, как можно заметить, гораздо короче. Вместо «заплатить этому скрипту с 5 ключами мультиподписи» эквивалент транзакции P2SH выглядит как «заплатить скрипту с этим хешем». Клиенту, проводящему платеж в пользу компании Мохаммеда, достаточно включить в него только этот намного более короткий скрипт выхода. Когда Мохаммед и его партнеры захотят потратить этот UTXO, им нужно будет представить первоначальный скрипт погашения (тот самый, хеш которого заблокировал UTXO) и подписи для его разблокировки. Например, так:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG>
```

Эти два скрипта объединяются в два этапа. Сначала скрипт погашения сверяется со скриптом выхода для проверки совпадения хешей:

```
<2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG> OP_HASH160 <хеш скрипта> OP_EQUAL
```

Если хеш скрипта погашения совпадает, то он выполняется:

```
<Sig1> <Sig2> 2 <PK1> <PK2> <PK3> <PK4> <PK5> 5 OP_CHECKMULTISIG
```

Адреса P2SH

Другой значимой особенностью P2SH является поддержка кодирования хеша скрипта в виде адреса, как это определено в BIP13. Адреса P2SH представляют собой кодировки 20-байтового хеша скрипта base58check, так же как биткойн-адреса представляют собой кодировки 20-байтового хеша открытого ключа в том же формате. В адресах P2SH используется префикс версии «5», поэтому в кодировке base58check они начинаются с «3».

Например, комплексный скрипт Мохаммеда, хешированный и перекодированный base58check как адрес P2SH, имеет вид `39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw`.

Теперь Мохаммед может дать этот «адрес» своим клиентам, и они смогут использовать практически любой биткойн-кошелек для совершения простого платежа, как и в случае с любым другим биткойн-адресом. Префикс 3 подскажет им, что это особый тип адреса, соответствующий скрипту, а не открытому ключу, но в остальном он работает точно так же, как и платеж на любой другой биткойн-адрес.

Адреса P2SH позволяют полностью скрыть все сложности, поэтому плательщик не увидит скрипт.

Преимущества P2SH

Возможности P2SH дают следующие преимущества по сравнению с прямым использованием комплексных скриптов на выходе:

- схожесть с изначальными старыми (legacy) адресами означает, что отправителю и его кошельку не нужна сложная техническая разработка для реализации P2SH;
- P2SH переносит бремя хранения данных длинного скрипта с выхода (который, помимо хранения в блокчейне, находится в наборе UTXO) на вход (хранится только в блокчейне);
- P2SH переносит бремя хранения данных длинного скрипта с настоящего времени (платеж) на будущее время (когда он будет потрачен);
- P2SH переносит стоимость комиссии за транзакцию с длинным скриптом от отправителя на получателя, который должен добавить длинный скрипт погашения для его расходования.

Скрипт погашения и проверка корректности

Невозможно поместить P2SH внутрь скрипта погашения P2SH, поскольку спецификации P2SH не допускают рекурсии. Технически можно включить `OP_RETURN` (см. «Выход записи данных (OP_RETURN)») в скрипт погашения, поскольку правила это не запрещают. Однако практической пользы от этого не будет, так

как выполнение OP_RETURN в процессе валидации приведет к признанию транзакции недействительной.

Обратите внимание: из-за того, что скрипт погашения не представлен в сети до того, как будет предпринята попытка потратить выход P2SH, создание выхода с хешем недействительного скрипта погашения не позволит его потратить. Расходная транзакция, включающая скрипт погашения, не будет принята из-за недействительного скрипта. Это создает определенные риски, поскольку в этом случае можно отправить биткойны на адрес P2SH, которые затем потратить не удастся.

Скрипты выхода P2SH содержат хеш скрипта погашения, который не дает никаких подсказок о содержании этого скрипта погашения. Выходные данные P2SH будут считаться действительными и принятыми даже с некорректным скриптом погашения. Можно совершенно случайно получить биткойны таким образом, что впоследствии их будет невозможно потратить.

Выход записи данных (OP_RETURN)

Распределенный и синхронизированный по времени блокчейн Биткойна обладает потенциалом для применения не только в сфере платежей. Многие разработчики пытались использовать преимущества языка скриптов транзакций, такие как безопасность и устойчивости системы, например, для создания приложений сферы цифровых нотариальных услуг. Первые попытки применить скриптовый язык Биткойна для этих целей сводились к созданию выходов транзакций с записью данных в блокчейн. Например, для записи обязательств по файлу таким образом, чтобы кто угодно, ссылаясь на эту транзакцию, мог доказать существование этого файла на определенную дату. Использование блокчейна Биткойна для хранения не связанных с биткойн-платежами данных является спорным вопросом. Многие считают это злоупотреблением и стремятся не допустить его. Другие рассматривают его как демонстрацию широких возможностей технологии блокчейна и поощряют подобные эксперименты. Противники включения в блокчейн данных неплатежного характера заявляют о недопустимости обременения полных узлов Биткойна расходами на дисковое хранение непрофильных для блокчейна данных. Более того, подобные транзакции могут создавать UTXO без возможности их расходования с использованием унаследованного биткойн-адреса в качестве 20-байтового поля свободной формы. Поскольку адрес используется для данных, он не соответствует секретному ключу, и полученный UTXO никогда не может быть потрачен; это фиктивный платеж. Поэтому такие никогда не расходуемые транзакции навсегда остаются в базе данных UTXO, что приводит к постоянному увеличению размера базы данных UTXO, то есть к ее «раздуванию».

В результате был достигнут компромисс, позволяющий скрипту выхода с оператором OP_RETURN добавлять данные неплатежного характера в выход транзакции. При этом, в отличие от создания «фальшивых» (fake) UTXO, оператор OP_RETURN создает однозначно *доказуемо нерасходуемый* (*provably unspendable*) выход, который не нужно хранить в наборе UTXO. Выходы OP_RETURN записыва-

ются в блокчейн, поэтому они занимают место на диске и способствуют увеличению размера блокчейна, однако они не хранятся в наборе UTXO и поэтому не «раздувают» полные узлы за счет более затратных операций с базой данных.

Скрипты OP_RETURN выглядят так:

```
OP_RETURN <data>
```

Часть данных зачастую представляет собой хеш, например выход алгоритма SHA256 (32 байта). Некоторые приложения для облегчения идентификации приложения добавляют перед данными префикс. Например, служба цифрового нотариального заверения Proof of Existence (https://proofofexistence.com/) использует 8-байтовый префикс DOCPROOF, который в шестнадцатеричном коде ASCII выглядит как 44 4f 43 50 52 4f 4f 46.

Следует помнить, что не существует никакого соответствующего OP_RETURN скрипта входа, который можно было бы использовать для «расходования» выхода OP_RETURN. Вся суть выхода OP_RETURN заключается в отсутствии возможности потратить деньги, заблокированные в этом выходе, и поэтому их не нужно держать в наборе UTXO в качестве потенциально возможных для расходов: выходы OP_RETURN являются *доказательно нерасходуемыми*. Выходы OP_RETURN обычно имеют нулевой баланс, потому что любые присвоенные такому выходу биткойны фактически навсегда потеряны. Если выход OP_RETURN используется в транзакции в качестве входа, механизм валидации скриптов остановит выполнение скрипта валидации и пометит транзакцию как недействительную. Выполнение OP_RETURN, по сути, приводит к возврату скрипта со значением FALSE и остановке. Таким образом, если случайно сослаться на выход OP_RETURN как на вход в транзакции, эта транзакция будет недействительной.

Ограничения времени блокировки транзакций

Время блокировки позволяет ограничить транзакцию от включения в блок до достижения им определенной высоты, но не препятствует расходованию средств в другой транзакции раньше этого срока. Это можно пояснить на следующем примере.

Алиса подписывает транзакцию, расходуя один из своих выходов на адрес Боба, и устанавливает время блокировки транзакции на 3 месяца в будущем. Алиса отправляет эту транзакцию на удержание Бобу. Алиса и Боб знают, что в результате:

- Боб не может передать транзакцию для погашения средств до истечения 3 месяцев;
- Боб может передать транзакцию через 3 месяца.

Однако:

- Алиса может создать конфликтующую транзакцию, потратив те же самые средства без блокировки по времени. Таким образом, Алиса может потратить те же UTXO до истечения 3 месяцев;
- у Боба нет гарантии, что Алиса не сделает этого.

Очень важно понимать особенности блокировки транзакций по времени. Единственная гарантия заключается в том, что Боб не сможет погасить предварительно подписанную транзакцию до истечения 3 месяцев. При этом нет никакой гарантии, что Боб получит средства. Один из способов обеспечить Бобу гарантию получения средств, но не возможность потратить их до истечения 3 месяцев – это наложить ограничение на блокировку по времени на сам UTXO в виде части скрипта, а не блокировки транзакции. Это можно сделать с помощью другой формы блокировки времени, называемой «проверка времени блокировки» (Check Lock Time Verify).

Проверка времени блокировки (OP_CLTV)

В декабре 2015 года в системе Биткойн появилась новая форма блокировки по времени в виде модификации софт-форка. Опираясь на спецификации BIP65, в язык скриптов был добавлен новый скриптовый оператор под названием OP_CHECK LOCKTIMEVERIFY (OP_CLTV). Оператор OP_CLTV предназначен для блокировки времени на выходе, а не по транзакции, как в случае с традиционной функцией блокировки по времени. Это обеспечивает дополнительную гибкость в использовании временных блокировок.

Проще говоря, в результате применения операции OP_CLTV к выходу расходование этого выхода становится ограниченным и может быть произведено только по истечении заданного времени.

Оператор OP_CLTV не заменяет блокировку по времени, а скорее ограничивает определенные UTXO так, что они могут быть потрачены только в будущей транзакции со временем блокировки, установленным на большее или равное значение. На входе OP_CLTV принимает единственный параметр, выраженный в виде числа в том же формате, что и время блокировки (либо высота блока, либо время эпохи в Unix). Суффикс VERIFY указывает на то, что OP_CLTV – это тип оператора, который останавливает выполнение скрипта при результате FALSE. В случае результата TRUE выполнение продолжается.

Для использования OP_CLTV необходимо вставить его в скрипт погашения выхода в транзакции, создающей этот выход. Например, если Алиса платит Бобу, он обычно принимает платеж в следующем скрипте P2SH:

```
<Bob's public key> OP_CHECKSIG
```

Чтобы заблокировать его на определенное время, скажем на 3 месяца, его скрипт P2SH должен выглядеть так:

```
<Bob's pubkey> OP_CHECKSIGVERIFY <now + 3 months> OP_CHECKLOCKTIMEVERIFY
```

где <now + 3 months> («сейчас + 3 месяца») – это высота блока или значение времени, рассчитанное на 3 месяца с момента майнинга транзакции: нынешняя высота блока + 12 960 (блоков), или актуальное время эпохи Unix + 7 760 000 (секунд).

Если Боб пытается потратить этот UTXO, он создает транзакцию со ссылкой на UTXO в качестве входа. Он использует свою подпись и открытый ключ в скрипте входа этого входа и устанавливает время блокировки транзакции

равным или превышающим время блокировки в установленном Алисой параметре OP_CHECKLOCKTIMEVERIFY. Затем Боб передает транзакцию в сеть Биткойн.

Транзакция Боба обрабатывается следующим образом. Если установленный Алисой параметр OP_CHECKLOCKTIMEVERIFY меньше или равен времени блокировки транзакции, выполнение скрипта продолжается (как если бы выполнялось *no operation* или код OP_NOP). В противном случае выполнение скрипта прекращается, а транзакция считается недействительной.

Еще более конкретно в BIP65 объясняется, что OP_CHECKLOCKTIMEVERIFY дает сбой и останавливает выполнение, если происходит одно из следующих событий:

- стек пуст;
- верхний элемент в стеке меньше 0;
- тип времени блокировки (высота или временная метка) верхнего элемента стека и поле времени блокировки не совпадают;
- верхний элемент стека больше, чем поле времени блокировки транзакции;
- поле sequence на входе равно 0xffffffff.

Конфликты блокировок по времени

Оба способа, OP_CLTV и блокировка по времени, используют один и тот же формат для описания временных блоков: либо высота блока, либо время, прошедшее в секундах с момента эпохи Unix. Крайне важно, чтобы при совместном использовании формат блокировки по времени совпадал с форматом OP_CLTV в выходах – они оба должны ссылаться либо на высоту блока, либо на время в секундах.

Это означает, что скрипт никогда не будет корректным, если ему нужно выполнить два разных вызова OP_CLTV, один из которых работает с высотой, а другой – со временем. Допустить такую ошибку при написании сложных скриптов несложно, поэтому следует тщательно тестировать свои скрипты в тестовой сети или использовать инструменты для предотвращения этой проблемы, например компилятор Miniscript.

Еще один момент заключается в том, что в любом из скриптов транзакции может использоваться только одна форма OP_CLTV. Если в скрипте для одного входа используется высота, а в другом скрипте для другого входа – время, то создать корректную транзакцию с использованием обоих входов не получится.

После выполнения, при условии успешного завершения OP_CLTV, предшествующий параметр остается верхним элементом стека и может быть сброшен с помощью OP_DROP для корректного выполнения последующих операций скрипта. По этой причине в скриптах часто встречается OP_CHECKLOCKTIMEVERIFY, за которым следует OP_DROP. OP_CLTV, как и OP_CSV (см. раздел «Относительные блокировки по времени»), не похожи на другие операции CHECKVERIFY в плане сохранения элементов в стеке, потому что добавившие их софт-форки переопределили уже существующие операции, не сбрасывающие элементы стека, и поэтому нужно сохранить характеристики этих предыдущих нулевых операций (NOP).

При использовании блокировки по времени в сочетании с OP_CLTV скрипт из раздела «Ограничения времени блокировки транзакций» меняется. Алиса немедленно отправляет свою транзакцию, закрепляя средства за ключом Боба. Алиса больше не может тратить деньги, но и Боб не может потратить их до истечения 3-месячного срока блокировки.

Благодаря внедрению возможности блокировки по времени непосредственно в язык скриптов OP_CLTV позволяет создавать очень интересные сложные скрипты.

Стандарт определен в протоколе BIP65 (OP_CHECKLOCKTIMEVERIFY) https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki.

Относительные блокировки по времени

Блокировка по времени и OP_CLTV являются *абсолютными временными блокировками* (*absolute timelocks*), поскольку в них задается абсолютная временная отметка. Две следующие функции, которые будут рассмотрены далее, являются *относительными временными блокировками* (*relative timelocks*), поскольку в них в качестве условия расходования средств указывается время, прошедшее с момента подтверждения вывода средств в блокчейн.

Относительные временные блокировки удобны наличием возможности наложить временное ограничение на одну транзакцию в зависимости от времени, прошедшего с момента подтверждения предыдущей транзакции. Другими словами, отсчет времени начинается только после записи UTXO в блокчейн. Эта возможность особенно полезна в каналах с двунаправленной передачей состояния и в сетях Lightning Networks (LN), как будет показано в разделе «Каналы платежей и каналы состояния».

Относительные временные блокировки, как и абсолютные, реализуются как на уровне транзакций, так и на уровне скриптов. Относительная блокировка по времени на уровне транзакций реализуется как правило консенсуса для значения поля sequence, задаваемого в каждом входе транзакции. Относительные блокировки по времени на уровне скриптов реализуются с помощью оператора OP_CHECKSEQUENCEVERIFY (OP_CSV).

Относительные временные блокировки применяются в соответствии со спецификациями BIP68 (Relative Lock-Time Using Consensus-Enforced Sequence Numbers https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki) и BIP112 (OP_CHECKSEQUENCEVERIFY https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki).

Протоколы BIP68 и BIP112 были добавлены в мае 2016 года в качестве софтфорка для обновления правил консенсуса.

Относительные блокировки по времени с OP_CSV

По аналогии с OP_CLTV и функцией блокировки по времени, для относительных временных блокировок применяется скриптовый операционный код с использованием значения поля sequence в скриптах. Этот оператор – OP_CHECKSEQUENCEVERIFY, обычно для краткости называют OP_CSV.

При использовании в скрипте UTXO оператор OP_CSV разрешает расходование средств только в той транзакции, значение sequence которой во входных данных больше или равно величине параметра OP_CSV. По сути, это ограничивает расходование UTXO до тех пор, пока не пройдет определенное количество блоков или секунд относительно времени майнинга UTXO.

Как и в случае с CLTV, значение в OP_CSV должно совпадать по формату с соответствующим значением поля sequence. Если OP_CSV задан в блоках, то и значение sequence должно быть таким же. Если OP_CSV задан в секундах, то и sequence должна быть такой же.

Скрипт, выполняющий несколько операций OP_CSV, должен использовать только один вид значения, либо по времени, либо по высоте блока. Смешивание приведет к созданию недействительного скрипта, который никогда не сможет быть использован, то есть к той же проблеме, что и в случае с OP_CLTV, описанной в разделе «Конфликты блокировок по времени». Однако OP_CSV позволяет включать в одну транзакцию любые два корректных входа, поэтому проблема взаимодействия между входами, возникающая при использовании OP_CLTV, не затрагивает OP_CSV.

Относительные временные блокировки с OP_CSV особенно удобны в случаях, когда несколько транзакций (в цепочке – chained) создаются и подписываются, но не рассылаются – то есть хранятся *вне блокчейна* (*offchain*). Дочерняя транзакция не может быть использована до момента, пока родительская транзакция не будет передана, добыта и выдержана в течение указанного в относительной временной блокировке времени. Одно из применений этого варианта показано в разделах «Платежные каналы и каналы состояний» и «Маршрутизированные платежные каналы (Lightning Network)».

Оператор OP_CSV подробно описан в спецификации BIP112, CHECKSEQUENCEVERIFY https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki.

Скрипты с контролем потока (условные предложения)

Одной из наиболее мощных функций языка Script Биткойна является система управления потоком (flow control), также известная как условные предложения (conditional clauses). Вероятно, вам знакомо использование управления потоком в различных языках программирования, где применяется конструкция IF…THEN…ELSE. Условные предложения Биткойна выглядят немного иначе, но, по сути, являются той же самой конструкцией.

На базовом уровне операции условий Биткойна позволяют создать скрипт, который имеет два способа разблокировки, в зависимости от результатов оценки логического условия TRUE/FALSE. Например, если x является TRUE, выполняется путь кода A, а при ELSE – путь кода B.

Кроме того, условные предложения Биткойна могут быть «вложенными» до бесконечности. То есть условное предложение может содержать внутри себя другое, которое содержит еще одно, и т. д. Управление потоком скриптов Бит-

койна можно использовать для построения очень сложных скриптов с сотнями возможных путей выполнения. Ограничений на уровень вложенности нет, но правила консенсуса накладывают ограничение на максимальный размер скрипта в байтах.

В Биткойне управление потоком осуществляется с помощью операторов OP_IF, OP_ELSE, OP_ENDIF и OP_NOTIF. Кроме того, условные выражения могут содержать булевы операторы, такие как OP_BOOLAND, OP_BOOLOR и OP_NOT.

На первый взгляд, скрипты управления потоком Биткойна могут показаться запутанными. Это связано с тем, что Bitcoin Script – стековый язык. Точно так же, как 1 + 1 выглядит «задом наперед», когда выражается как 1 1 OP_ADD, условия управления потоком в Биткойне также выглядят «задом наперед».

В большинстве традиционных (процедурных) языков программирования управление потоком выглядит следующим образом:

```
if (условие):
  код, который будет выполняться, если условие истинно (true)
else:
  код, который будет выполняться, если условие ложно (false)
endif
  код, выполняемый в любом случае
```

В стековых языках, таких как Bitcoin Script, логическое условие идет перед IF, из-за чего оно выглядит «задом наперед»:

```
условие
IF
  код, который будет выполняться, если условие истинно (true)
OP_ELSE
  код, который будет выполняться, если условие ложно (false)
OP_ENDIF
  код, выполняемый в любом случае
```

При чтении скриптов Bitcoin Script помните, что оцениваемое условие идет перед операцией IF.

Условные предложения с оператором VERIFY

Другой формой условия в языке Script Биткойна является любая операция, которая заканчивается на VERIFY. Суффикс VERIFY означает, что если вычисленное условие не является TRUE, выполнение скрипта немедленно прекращается, а транзакция считается недействительной.

В отличие от условия IF, которое предлагает альтернативные пути выполнения, суффикс VERIFY действует как *защитное условие* (*guard clause*), продолжая выполнение только при соблюдении предварительного условия.

Например, следующий скрипт требует подписи Боба и предварительного образа (секрета), который создает определенный хеш. Для разблокировки необходимо выполнить оба условия:

```
OP_HASH160 <expected hash> OP_EQUALVERIFY <Bob's Pubkey> OP_CHECKSIG
```

Для этого Боб должен представить действительный предварительный образ и подпись:

```
<Bob's Sig> <hash pre-image>
```

Не предоставив предварительный образ, Боб не сможет добраться до той части скрипта, которая проверяет наличие его подписи.

Вместо этого скрипт можно написать с помощью OP_IF:

```
OP_HASH160 <expected hash> OP_EQUAL
OP_IF
   <Bob's Pubkey> OP_CHECKSIG
OP_ENDIF
```

Данные для аутентификации Боба идентичны:

```
<Bob's Sig> <hash pre-image>
```

Скрипт с OP_IF делает то же самое, что и использование оператора с суффиксом VERIFY; они оба работают как защитные условия. Однако конструкция VERIFY более эффективна, в ней используется на две операции меньше.

Итак, где использовать VERIFY, а где OP_IF? Если все, что нужно сделать, – это добавить предварительное условие (*защитное предложение – guard clause*), то лучше использовать VERIFY. Если же необходимо иметь более одного пути выполнения (управление потоком), то потребуется условие управления потоком OP_IF...OP_ELSE.

Использование управления потоком в скриптах

Очень часто контроль потока в Bitcoin Script используется для создания скрипта с несколькими путями выполнения, каждый из которых отличается способом погашения UTXO.

Рассмотрим простой пример, в котором есть две подписи, от Алисы и Боба, и любая из них может произвести погашение. При использовании мультиподписи это будет выражено в виде скрипта с несколькими подписями вида 1-of-2. Для наглядности то же самое можно сделать с помощью условия OP_IF:

```
OP_IF
 <Alice's Pubkey>
OP_ELSE
 <Bob's Pubkey>
OP_ENDIF
OP_CHECKSIG
```

Глядя на этот скрипт погашения, можно задаться вопросом: «Где же условие? Ведь перед условием IF ничего нет!»

Условие не является частью скрипта. Вместо этого условие будет предлагаться в момент совершения расходов, позволяя Алисе и Бобу «выбрать» желаемый путь выполнения:

```
<Alice's Sig> OP_TRUE
```

Здесь OP_TRUE в конце служит условием (TRUE), которое заставит условие OP_IF выполнить первый путь погашения. Это условие помещает в стек открытый ключ, для которого у Алисы есть подпись. Операция OP_TRUE, также известная как OP_1, поместит в стек число 1.

Чтобы Боб смог его погасить, ему придется выбрать второй путь выполнения в OP_IF, указав значение FALSE. Операция OP_FALSE, также известная как OP_0, помещает в стек пустой массив байтов:

```
<Bob's Sig> OP_FALSE
```

Скрипт входа Боба заставляет условие OP_IF выполнить второй скрипт (OP_ELSE), который требует подписи Боба.

Поскольку условия OP_IF могут быть вложенными, можно создать «лабиринт» путей выполнения. Скрипт входа может предоставить «карту» для выбора фактического пути выполнения:

```
OP_IF
  subscript A
OP_ELSE
  OP_IF
    subscript B
  OP_ELSE
    subscript C
  OP_ENDIF
OP_ENDIF
```

В этом сценарии есть три пути выполнения (subscript A, subscript B и subscript C). Скрипт входа предоставляет путь в виде последовательности значений TRUE или FALSE. Например, чтобы выбрать путь subscript B, скрипт входа должен заканчиваться на OP_1 OP_0 (TRUE, FALSE). Эти значения будут помещены в стек таким образом, чтобы второе значение (FALSE) оказалось на вершине стека. Внешнее условие OP_IF выводит значение FALSE и выполняет первое условие OP_ELSE. Затем значение TRUE перемещается на вершину стека и оценивается внутренним (вложенным) OP_IF, выбирая путь выполнения B.

Используя эту конструкцию, можно создавать скрипты погашения с десятками или сотнями путей выполнения, каждый из которых предлагает свой способ погашения UTXO. Чтобы потратить деньги, нужно создать скрипт входа, который перемещается по пути выполнения, записывая в стек соответствующие значения TRUE и FALSE в каждой точке управления потоком.

Пример сложного скрипта

В этом разделе многие концепции из данной главы будут рассмотрены на одном примере.

Мохаммед, владелец компании из Дубая, занимается импортом/экспортом; он хочет создать корпоративный счет основного капитала с гибкими правилами. Созданная им схема требует различных уровней авторизации в зависимости от временных интервалов. Участниками схемы с несколькими подпися-

ми являются Мохаммед (Mohammed), два его партнера – Саид (Saeed) и Заира (Zaira), а также юрист (lawyer) их компании. Три партнера принимают решения по принципу большинства, поэтому двое из трех должны дать свое согласие. Однако на случай проблем с ключами они хотят, чтобы их юрист мог вернуть средства с помощью одной из трех партнерских подписей. Наконец, если все партнеры будут отсутствовать или станут недееспособными на какое-то время, они хотят, чтобы юрист мог управлять счетом напрямую, получив доступ к записям операций по счету основного капитала.

Пример 7.1 – это скрипт погашения (redeem script), который Мохаммед разработал для решения этой задачи (номера строк снабжены префиксами).

Пример 7.1 ❖ Изменяемая мультиподпись с блокировкой по времени

```
01  OP_IF
02    OP_IF
03      2
04    OP_ELSE
05      <30 days> OP_CHECKSEQUENCEVERIFY OP_DROP
06      <Lawyer's Pubkey> OP_CHECKSIGVERIFY
07      1
08    OP_ENDIF
09    <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's Pubkey> 3 OP_CHECKMULTISIG
10  OP_ELSE
11    <90 days> OP_CHECKSEQUENCEVERIFY OP_DROP
12    <Lawyer's Pubkey> OP_CHECKSIG
13  OP_ENDIF
```

Скрипт Мохаммеда реализует три пути выполнения с помощью вложенных условий управления потоком OP_IF…OP_ELSE.

На первом пути выполнения этот скрипт работает как простая мультиподпись 2-of-3 с тремя партнерами. Этот путь выполнения состоит из строк 3 и 9. Строка 3 устанавливает кворум мультиподписи на 2 (2-of-3). Этот путь выполнения можно выбрать, поставив OP_TRUE OP_TRUE в конце скрипта входных данных:

```
OP_0 <Mohammed's Sig> <Zaira's Sig> OP_TRUE OP_TRUE
```

Код OP_0 в начале этого скрипта входа вызван странностью в OP_CHECKMULTISIG, который извлекает из стека избыточное значение. Это дополнительное значение игнорируется OP_CHECKMULTISIG, но оно должно присутствовать, иначе скрипт не будет выполнен. Загрузка пустого байтового массива с помощью OP_0 является обходным решением этой странности, как описано в разделе «Нестандартное выполнение CHECKMULTISIG».

Второй путь выполнения можно использовать только по истечении 30 дней с момента создания UTXO. В этот момент требуется подпись юриста и одного из трех партнеров (мультиподпись 1-of-3). Это обеспечивается строкой 7, которая устанавливает кворум для мультиподписи на 1. Чтобы выбрать этот путь исполнения, скрипт входа должен завершаться OP_FALSE OP_TRUE:

```
OP_0 <Saeed's Sig> <Lawer's Sig> OP_FALSE OP_TRUE
```

Почему OP_FALSE OP_TRUE? Разве это не обратная ситуация? FALSE помещается в стек, а TRUE помещается на его вершину. Таким образом, TRUE первым выталкивается из стека первым оператором OP_IF.

Наконец, третий путь выполнения позволяет адвокату тратить средства самостоятельно, но только через 90 дней. Чтобы выбрать этот путь выполнения, скрипт входа должен заканчиваться на OP_FALSE:

```
<Lawyer's Sig> OP_FALSE
```

Попробуйте прогнать скрипт на бумаге, чтобы увидеть, как он ведет себя в стеке.

Примеры выхода сегрегированного свидетеля и транзакций

Рассмотрим некоторые из приведенных примеров транзакций и посмотрим, как они изменятся при использовании сегрегированного (изолированного) свидетеля (segregated witness). Сначала выясним, как можно осуществить платеж P2PKH с помощью программы segregated witness. Затем обратимся к эквиваленту segregated witness для скриптов P2SH. И наконец, узнаем, как оба предыдущих варианта программы segregated witness могут быть встроены в скрипт P2SH.

Платеж pay to witness public key hash (P2WPKH)

Для начала рассмотрим пример скрипта выхода P2PKH:

```
OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
OP_EQUALVERIFY OP_CHECKSIG
```

С помощью механизма segregated witness Алиса создает скрипт P2WPKH. Если этот скрипт подтвердит тот же открытый ключ, это будет выглядеть так:

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Как можно видеть, скрипт выхода P2WPKH намного проще, чем его эквивалент P2PKH. Он состоит из двух значений, которые помещаются в стек оценки скрипта. Для старого типа (без segwit) клиента Биткойна эти два значения будут выглядеть как выход, который может провести любой желающий. Для более нового клиента с поддержкой segwit первое число (0) интерпретируется как номер версии (версия свидетеля), а вторая часть (20 байт) является *программой свидетеля* (*witness program*). Эта 20-байтовая программа свидетеля не что иное, как хеш открытого ключа, как в скрипте P2PKH.

Теперь рассмотрим соответствующую транзакцию, которую Боб использует для расходования этого выхода. В исходном скрипте тратящая транзакция должна была содержать подпись на входе:

```
[...]
"vin" : [
  "txid": "abcdef12345...",
  "vout": 0,
  "scriptSig": "<Bob's scriptSig>",
]
[...]
```

При этом, чтобы потратить выход P2WPKH, транзакция не подписывает этот вход. Вместо этого транзакция Боба имеет пустой входной скрипт и включает структуру свидетеля:

```
[...]
"vin" : [
  "txid": "abcdef12345...",
  "vout": 0,
  "scriptSig": "",
]
[...]
"witness": "<Bob's witness structure>"
[...]
```

Конструкция кошелька P2WPKH

Крайне важно отметить, что программы свидетелей P2WPKH могут быть созданы только получателем, а не преобразованы отправителем из известного открытого ключа, скрипта P2PKH или адреса. Отправитель не имеет возможности узнать, есть ли у кошелька получателя возможность создавать транзакции segwit и тратить выходы P2WPKH.

Кроме того, данные выхода P2WPKH должны быть созданы из хеша сжатого открытого ключа. Несжатые открытые ключи не являются стандартом в segwit и могут быть однозначно отключены в будущем софт-форке. Если хеш в P2WPKH получен из несжатого открытого ключа, он может оказаться нерасходуемым, и вы лишитесь средств. Выходные данные P2WPKH должны создаваться кошельком получателя платежа путем получения сжатого открытого ключа из его секретного ключа.

P2WPKH должен быть создан получателем при помощи преобразования сжатого открытого ключа в хеш P2WPKH. Ни отправитель, ни кто-либо другой никогда не должен преобразовывать скрипт P2PKH, биткойн-адрес или несжатый открытый ключ в скрипт свидетеля P2WPKH. Как правило, отправитель обязан осуществлять передачу получателю только тем способом, который указал сам получатель.

Платеж pay to witness script hash (P2WSH)

Второй тип программы свидетеля segwit v0 соответствует скрипту P2SH. Этот тип скрипта был рассмотрен в разделе «Скрипт Pay to Script Hash (P2SH)». В том примере P2SH использовался компанией Мохаммеда для реализации скрипта с несколькими подписями. Платежи в адрес компании Мохаммеда были закодированы следующим скриптом:

```
OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL
```

Этот скрипт P2SH ссылается на хеш скрипта погашения, который определяет требование 2-of-3 мультиподписи для расходования средств. Чтобы потратить эти средства, компания Мохаммеда должна представить скрипт погашения (хеш которого совпадает с хешем скрипта в выводе P2SH) и подписи, необходимые для соответствия этому скрипту погашения, и все это во входе транзакции:

```
[...]
"vin" : [
  "txid": "abcdef12345...",
  "vout": 0,
  "scriptSig": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 OP_CHECKMULTISIG>",
]
```

Теперь посмотрим, как весь этот пример можно обновить до segwit v0. Если бы клиенты Мохаммеда использовали segwit-совместимый кошелек, они бы провели платеж, создав вывод P2WSH следующего вида:

```
0 a9b7b38d972cabc7961dbfbcb841ad4508d133c47ba87457b4a0e8aae86dbb89
```

Как и в случае с P2WPKH, эквивалентный скрипт сегрегированного свидетеля намного проще. Он уменьшает избыточность шаблона, характерную для скриптов P2SH. Вместо этого скрипт выхода сегрегированного свидетеля состоит из двух значений, помещаемых в стек: версии свидетеля (0) и 32-байтового хеша SHA256 скрипта свидетеля (программы свидетеля).

Если в P2SH используется 20-байтовый хеш `RIPEMD160(SHA256(script))`, то в программе свидетеля P2WSH применяется 32-байтовый хеш `SHA256(script)`. Такое различие в выборе алгоритма хеширования сделано намеренно для повышения безопасности P2WSH в отдельных случаях использования (128 бит защиты в P2WSH против 80 бит защиты в P2SH). Подробнее см. в разделе «Коллизионные атаки на P2SH».

Компания Мохаммеда может потратить выход P2WSH, представив корректный скрипт свидетеля и достаточное количество подписей для его выполнения. Скрипт свидетеля и подписи будут включены в структуру свидетеля. Никаких данных в скрипт входа помещать не нужно, поскольку это нативная программа свидетеля, которая не использует унаследованное поле скрипта входа:

```
[...]
"vin" : [
  "txid": "abcdef12345...",
  "vout": 0,
  "scriptSig": "",
]
[...]
"witness": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 OP_CHECKMULTISIG>"
[...]
```

Различие между P2WPKH и P2WSH

В предыдущих двух разделах были представлены два типа программ свидетелей: «оплата по хешу открытого ключа свидетеля» (pay to witness public key hash, P2WPKH)» и «оплата по хешу скрипта свидетеля» (pay to witness script hash, P2WSH). Оба типа программ свидетелей состоят из одного и того же номера версии, за которым следует ввод данных. Они выглядят очень похоже, но имеют разную интерпретацию: одна трактуется как хеш открытого ключа, который подтверждается подписью, а другая – как хеш скрипта, который подтверждается скриптом свидетеля. Критическое различие между ними заключается в длине программы свидетеля:

- программа свидетеля в P2WPKH равна 20 байтам;
- программа свидетеля в P2WSH равна 32 байтам.

Это единственное отличие, по которому полные узлы могут отличить два типа программ свидетелей. По длине хеша узел может определить вид программы свидетеля – P2WPKH или P2WSH.

Обновление до Segregated Witness

Как видно из предыдущих примеров, переход к сегрегированному свидетелю состоит из двух этапов. Сначала кошельки должны создать выходы типа segwit. Затем эти выходы могут быть потрачены кошельками, которые умеют создавать транзакции с segregated witness. В приведенных примерах кошелек Алисы способен создавать выходы, оплачивая скрипты выхода segregated witness. Кошелек Боба также знает о segwit и может расходовать эти средства.

Реализация segregated witness была сделана как обратно совместимое обновление, где *могут уживаться старые и новые клиентские программы*. Разработчики кошельков независимо друг от друга обновили программы кошельков для добавления функций segwit. Устаревшие P2PKH и P2SH продолжают использоваться в кошельках, не прошедших модернизацию. В результате остаются два основных сценария, которые рассматриваются в следующем разделе:

- возможность кошелька отправителя с поддержкой segwit совершить платеж на кошелек получателя с поддержкой segwit-транзакций;
- способность кошелька отправителя с функцией segwit распознавать и видеть отличия между получателями с поддержкой segwit и без нее *по их адресам*.

Внедрение сегрегированного свидетеля в P2SH

Представим, что кошелек Алисы не обновлен до поддержки segwit, а кошелек Боба уже обновлен и может обрабатывать транзакции segwit. Алиса и Боб могут использовать старые выходы без segwit. Но Боб, скорее всего, захочет использовать segwit ради снижения комиссии за транзакции, используя преимущества снижения стоимости структуры свидетелей.

В этом случае кошелек Боба может создать адрес P2SH со скриптом segwit внутри. Кошелек Алисы может проводить на него платежи без какого-либо представления о segwit. Затем кошелек Боба может потратить этот платеж с помощью транзакции segwit, используя преимущества segwit и снижая комиссию за транзакции.

Обе формы скриптов свидетелей, P2WPKH и P2WSH, могут быть встроены в адрес P2SH. Первый вариант обозначается как вложенный P2WPKH, а второй – как вложенный P2WSH.

Вложенный платеж pay to witness public key hash (Nested P2WPKH)

Первая рассматриваемая форма скрипта выхода – вложенный (nested) P2WPKH. Это программа pay to witness public key hash witness, встроенная в скрипт pay to script hash, поэтому кошелек без segwit может оплачивать скрипт выхода.

Кошелек Боба создает программу свидетеля P2WPKH с открытым ключом Боба. Затем эта программа свидетеля хешируется, и полученный хеш кодируется как скрипт P2SH. Скрипт P2SH преобразуется в биткойн-адрес, который начинается с цифры «3», как было показано в разделе «Скрипт Pay to Script Hash (P2SH)».

Кошелек Боба начинает работу с версией свидетеля P2WPKH и рассмотренной ранее программой свидетеля:

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Данные включают в себя версию свидетеля и 20-байтовый хеш открытого ключа Боба.

Затем кошелек Боба хеширует полученные данные, сначала с помощью SHA256, потом с помощью RIPEMD-160, создавая еще один 20-байтовый хеш. Затем хеш скрипта погашения преобразуется в биткойн-адрес. Наконец, кошелек Алисы может совершить платеж на `37Lx99uaGn5avKBxiW26HjedQE3LrDCZru`, как и на любой другой биткойн-адрес.

Чтобы заплатить Бобу, кошелек Алисы заблокирует выход с помощью скрипта P2SH:

```
OP_HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b OP_EQUAL
```

Несмотря на отсутствие поддержки segwit в кошельке Алисы, созданный ею платеж может быть потрачен Бобом с помощью segwit-транзакции.

Вложенный платеж pay to witness script hash (Nested P2WSH)

Аналогичным образом программа свидетеля P2WSH для мультиподписи или другого сложного скрипта может быть встроена в скрипт и адрес P2SH, что позволит любому кошельку проводить платежи, совместимые с segwit.

Как было показано в разделе «Платеж pay to witness script hash (P2WSH)», компания Мохаммеда использует платежи с помощью segregated witness для мультиподписи скриптов. Чтобы любой клиент мог заплатить его компании, независимо от наличия обновления кошелька до segwit, кошелек Мохаммеда может встроить программу свидетеля P2WSH в скрипт P2SH.

Сначала кошелек Мохаммеда хеширует скрипт свидетеля с помощью SHA256 (только один раз), получая в результате хеш:

```
9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Затем хешированный скрипт свидетеля превращается в программу свидетеля P2WSH с префиксом версии:

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Потом сама программа свидетеля хешируется с помощью SHA256 и RIPEMD-160, в результате чего получается новый 20-байтовый хеш:

```
86762607e8fe87c0c37740cddee880988b9455b2
```

Далее кошелек создает из этого хеша биткойн-адрес P2SH:

```
3Dwz1MXhM6EfFoJChHCxh1jWHb8GQqRenG.
```

Теперь клиенты Мохаммеда могут совершать платежи на этот адрес, даже если у них отсутствует поддержка segwit. Чтобы отправить платеж Мохаммеду, кошелек заблокирует выход с помощью следующего скрипта P2SH:

```
OP_HASH160 86762607e8fe87c0c37740cddee880988b9455b2 OP_EQUAL
```

Затем компания Мохаммеда может создавать транзакции segwit для проведения этих платежей с использованием всех преимуществ segwit, включая более низкие комиссии за транзакции.

Деревья альтернативных скриптов (Merklized Alternative Script Trees, MAST)

С помощью `OP_IF` можно авторизовать различные условия расходования средств, но у этого подхода есть несколько нежелательных аспектов.

Вес (стоимость)
: Каждое добавленное условие увеличивает размер скрипта, повышая вес (weight) транзакции и размер комиссии, которую нужно будет заплатить для расходования биткойнов, защищенных этим скриптом.

Ограниченный размер
: Несмотря на готовность платить за лишние условия, существует ограничение на максимальное количество допустимой в скрипте информации. Даже если бы удалось создать скрипт размером с целый блок, он все равно содержал бы около 20 000 полезных ветвей. Это много для простых платежей, но ничтожно мало по сравнению с другими вариантами использования Биткойна.

Недостаточная конфиденциальность
: Каждое добавленное в скрипт условие получает огласку при расходовании биткойнов, защищенных этим скриптом. Например, адвокат и деловые партнеры Мохаммеда смогут видеть весь скрипт из примера 7.1, когда кто-нибудь будет совершать расходы с его помощью. Это означает, что их адвокат, даже если он не участвует в подписании, сможет отслеживать все их транзакции.

Однако в Биткойне уже используется структура данных под названием «дерево Меркла» (merkle tree), которая дает возможность проверить принадлежность элемента к множеству без необходимости идентификации всех остальных элементов этого множества.

Подробнее о деревьях Меркла рассказывается в разделе «Деревья Меркла», но суть заключается в том, что элементы нужного нам набора данных (например, условия авторизации любой длины) могут быть переданы в хеш-функцию для создания короткого обязательства. Это называется *листом* дерева Меркла (*leaf* of the merkle tree). Затем каждый из этих листьев объединяется с другим листом и снова хешируется, создавая обязательство по листьям. Это называется обязательством по *ветвям* (*branch* commitment). Таким же образом можно создать обязательство для пары ветвей. Этот шаг повторяется для всех ветвей, пока не останется только один идентификатор, называемый корнем Меркла (*merkle root*).

Используя скрипт из примера 7.1, можно построить дерево Меркла для каждого из трех условий авторизации, как показано на рис. 7.5.

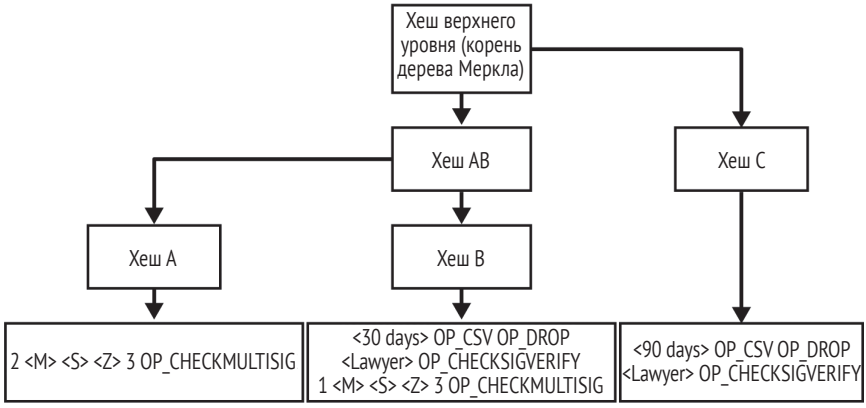


Рис. 7.5 ❖ MAST с тремя подписями

Теперь можно создать компактное доказательство принадлежности, которое доказывает, что конкретное условие авторизации является элементом дерева Меркла, не раскрывая никаких подробностей о других его элементах. См. рис. 7.6, где отмечено, что заштрихованные узлы могут быть вычислены из других предоставленных пользователем данных, поэтому их не нужно указывать в момент проведения платежа.

Дайджесты хешей, используемые для создания обязательств, содержат по 32 байта, поэтому доказательство авторизации (с помощью дерева Меркла и определенных условий) и аутентификации (с помощью подписей) для расходов, представленных на рис. 7.6, занимает 383 байта. Для сравнения: та же самая операция расходования средств без использования дерева Меркла (то есть с предоставлением всех возможных условий авторизации) требует 412 байт.

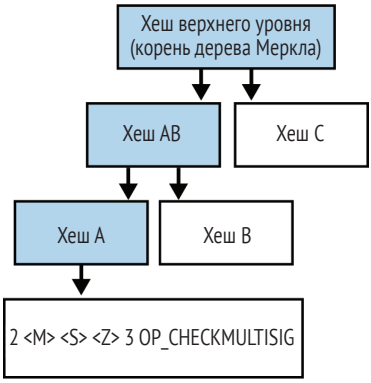


Рис. 7.6 ❖ Доказательство принадлежности к MAST для одного из субскриптов

Экономия 29 байт (7 %) в этом примере не дает полного представления о потенциальной экономии. Двоичная природа дерева Меркла означает, что при удвоении количества элементов в наборе (в данном случае условий авторизации) потребуется только дополнительное обязательство размером 32 байта. В данном случае, при наличии трех условий, необходимо использовать три обязательства (одно из них – корень дерева Меркла, который должен быть включен в состав данных авторизации). При тех же затратах можно было бы использовать и четыре обязательства. Дополнительное обязательство даст до восьми условий. При использовании всего 16 обязательств – 512 байт обязательств – можно получить более 32 000 условий авторизации, что гораздо больше, чем может быть эффективно использовано в целом блоке транзакций, заполненном операторами `OP_IF`. При 128 обязательствах (4096 байт) количество теоретически возможных условий намного превышает количество условий, которые могут создать все компьютеры в мире.

Как правило, не все условия авторизации используются с одинаковой вероятностью. В примере с Мохаммедом и его партнерами предполагается, что они будут часто тратить свои деньги; а условия задержки по времени существуют только на случай, если что-то пойдет не так. Учитывая это, можно изменить структуру дерева, как показано на рис. 7.7.

Теперь нужно предоставить только два обязательства для обычного случая (экономия 32 байта), хотя для менее распространенных ситуаций по-прежнему требуется три обязательства. Когда известны (или можно предположить) вероятности использования различных условий авторизации, можно использовать алгоритм Хаффмана (Huffman algorithm) для размещения их в максимально эффективном дереве. Подробности см. в BIP341.

Независимо от способа построения дерева, в предыдущих примерах можно увидеть, что раскрываются только фактические используемые условия авторизации. Остальные условия остаются в тайне. Также остается скрытым количество условий: в дереве может быть одно условие или триллион условий – никто не сможет определить это, просматривая только данные по одной транзакции в цепочке.

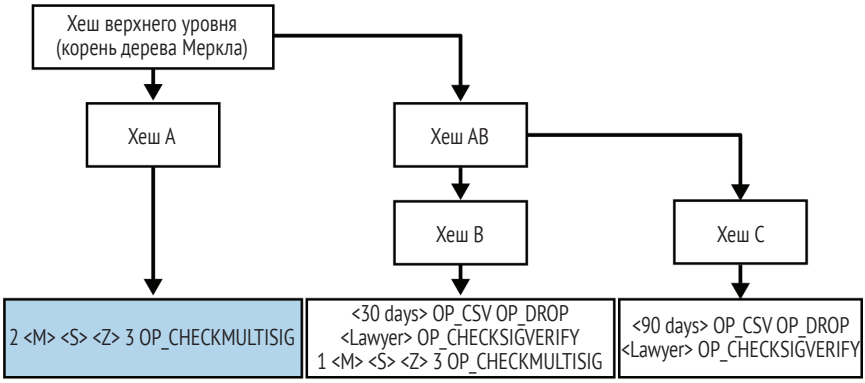


Рис. 7.7 ❖ MAST с наиболее ожидаемым скриптом в наилучшей позиции

За исключением небольшого усложнения Биткойна, у MAST нет существенных недостатков для него. Кроме того, было два серьезных предложения, BIP114 и BIP116, прежде чем был найден улучшенный вариант, который будет рассмотрен в разделе «Taproot».

MAST против MAST

Самая ранняя концепция того, что сейчас называют *MAST* в Биткойне, представляла собой *мерклизированные абстрактные синтаксические деревья* (*merklized abstract syntax trees*). В абстрактном синтаксическом дереве (abstract syntax tree, AST) каждое условие в скрипте создает новую ветвь, как показано на рис. 7.8.

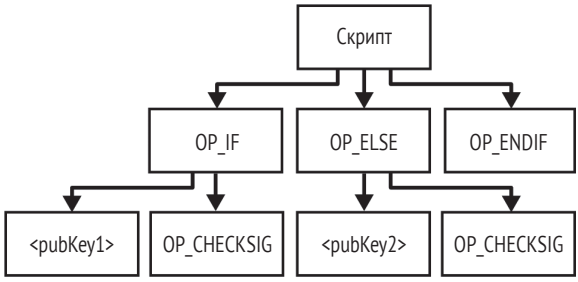


Рис. 7.8 ❖ Абстрактное синтаксическое дерево (AST) для скрипта

Структуры AST широко используются программами для анализа и оптимизации кода других программ, например компиляторами. Мерклизированное AST фиксирует каждую часть программы и дает возможность использовать функции, описанные в разделе «Деревья альтернативных скриптов (Merklized Alternative Script Trees, MAST)». Однако это потребовало бы раскрытия как минимум одного 32-байтового дайджеста для каждой отдельной части программы, что не очень эффективно для большинства программ блокчейна в плане использования пространства.

То, что сегодня в Биткойне принято называть *MAST*, – это *мерклизированные альтернативные деревья скриптов* (*merklized alternative script trees*), этакая псевдо-

аббревиатура (бэкроним), придуманная разработчиком Энтони Таунсом (Anthony Towns). Альтернативное дерево скриптов – это набор скриптов, каждый из которых является завершенным сам по себе и в котором может быть выбран только один, что делает их альтернативными друг другу, как показано на рис. 7.9.

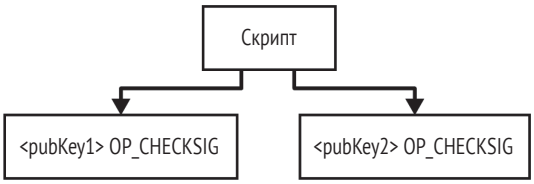


Рис. 7.9 ❖ Альтернативное дерево скриптов

Альтернативные деревья скриптов требуют раскрытия только одного 32-байтового дайджеста для каждого уровня глубины между выбранным пользователем скриптом и корнем дерева. Для большинства скриптов это гораздо более эффективное использование пространства в блокчейне.

Платеж Pay to Contract (P2C)

Как было показано в разделе «Получение открытого дочернего ключа», математика криптографии на эллиптических кривых (elliptic curve cryptography, ECC) позволяет Алисе использовать секретный ключ для выведения открытого ключа, который она передает Бобу. Он может добавить к этому открытому ключу произвольное значение для создания производного открытого ключа. Если он передаст это произвольное значение Алисе, она сможет добавить его к своему секретному ключу для получения секретного ключа от производного открытого ключа. Проще говоря, Боб может создавать дочерние открытые ключи, для которых соответствующие секретные ключи может создать только Алиса. Такая возможность полезна для восстановления иерархических детерминированных (Hierarchical Deterministic, HD) кошельков по стандарту BIP32, но может пригодиться и в других случаях.

Представим, что Боб хочет что-то купить у Алисы, но ему также хотелось бы иметь впоследствии возможность подтвердить факт покупки в случае возникновения споров. Алиса и Боб договариваются о названии продаваемого товара или услуги (например, «Эпизод № 123 подкаста Алисы») и преобразуют это описание в число путем его хеширования и интерпретации дайджеста хеша как числа. Боб добавляет это число к открытому ключу Алисы и производит оплату. Этот процесс называется «*настройка* (*твик*) *ключа*» (*key tweaking*), а полученное число называют «*твик*» (*tweak*).

Алиса может потратить средства, подстроив свой секретный ключ с помощью того же числа («твика»).

Позже Боб может доказать кому угодно размер своей платы Алисе, раскрыв ее базовый ключ и использованное ими описание. Любой может проверить, что открытый ключ, которым была произведена оплата, равен базовому клю-

чу плюс хеш-обязательство описания. Если Алиса признает свой ключ, значит, она получила оплату. Если Алиса потратила деньги, это еще раз доказывает, что она была знакома с описанием в момент подписания транзакции, поскольку она могла создать действительную подпись для «твикнутого» (настроенного) открытого ключа, только если знала «твик» (описание).

Если ни Алиса, ни Боб не решили публично раскрыть использованное ими описание, расчет между ними выглядит как любой другой платеж. Никакой потери конфиденциальности при этом не происходит.

Поскольку механизм «*оплата по контракту*» (*Pay to Contract*, *P2C*) по определению является секретным, никто не может знать, как часто он используется по своему прямому назначению – теоретически его можно использовать при каждом платеже, хотя такой вариант кажется маловероятным. Однако сегодня P2C широко используется в несколько иной форме, которая будет рассмотрена в разделе «Платеж Taproot».

Мультиподписи без скриптов и подписи с порогом

В разделе «Скриптовые мультиподписи» были описаны скрипты, требующие наличия подписей от нескольких ключей. Однако есть и другой способ запросить взаимодействие с несколькими ключами, который также по непонятной причине называют «*мультиподпись*» (*multisignature*). Чтобы различать эти два типа в данном разделе, в дальнейшем будем называть версию с использованием операций типа `OP_CHECKSIG` *скриптовыми мультиподписями* (*script multisignatures*), а другую версию – м*ультиподписями без скриптов* (*scriptless multisignatures*).

Мультиподписи без скриптов предполагают создание каждым из участников своего собственного секрета аналогично тому, как создается секретный ключ. Этот секрет будем называть *частичным секретным ключом* (*partial private key*), хотя стоит отметить, что он имеет ту же длину, что и обычный полный секретный ключ. Из частичного секретного ключа каждый участник получает *частичный открытый ключ* (*partial public key*) с помощью того же алгоритма, что и для обычных открытых ключей, который был описан в разделе «Открытые ключи». Каждый участник делится своими частичными открытыми ключами со всеми остальными участниками, а затем объединяет все ключи вместе для создания открытого ключа мультиподписи без скрипта.

Этот комбинированный открытый ключ (combined public key) выглядит так же, как и любой другой открытый ключ Биткойна. Третья сторона не сможет отличить многосторонний открытый ключ от обычного ключа, сгенерированного одним пользователем.

Чтобы потратить биткойны, защищенные открытым ключом мультиподписи без скриптов, каждый участник создает частичную подпись. Затем частичные подписи объединяются для создания обычной полной подписи. Существует множество различных способов создания и сочетания частичных подписей; подробнее эта тема будет рассмотрена в главе 8. Как и открытые ключи для мультиподписей без скриптов, подписи, созданные в результате этого процесса,

выглядят как обычные подписи Биткойна. Посторонние лица не смогут определить, была подпись создана одним человеком или сообща миллионом людей.

Мультиподписи без скриптов компактнее и более конфиденциальны, чем мультиподписи со скриптами. Для мультиподписей со скриптами количество байтов в транзакции увеличивается с каждым задействованным ключом и подписью. В случае мультиподписей без скриптов размер остается постоянным – миллион участников, каждый из которых предоставляет свой частичный ключ и частичную подпись, помещают в транзакцию ровно столько же данных, сколько один человек с одним ключом и одной подписью. С конфиденциальностью дело обстоит аналогичным образом. Поскольку каждый новый ключ или подпись добавляет данные в транзакцию, мультиподписи со скриптами раскрывают данные о количестве используемых ключей и подписей, что может облегчить выяснение информации о транзакциях, созданных той или иной группой участников. Поскольку каждая мультиподпись без скриптов выглядит как любая другая мультиподпись без скриптов и любая одиночная подпись, никакой утечки конфиденциальных данных не происходит.

У мультиподписи без скрипта есть два недостатка. Первый заключается в необходимости использования всех известных безопасных алгоритмов Биткойна для их создания, что требует большего количества этапов взаимодействия или более тщательного управления процессами изменения состояния, чем скрипты мультиподписей. Это может оказаться непростой задачей в случаях, когда подписи генерируются аппаратными устройствами подписи, практически лишенными функций сохранения состояния, с физическим распределением ключей. Например, если вы храните аппаратное устройство для подписи в банковском сейфе, то для создания мультиподписи со скриптом потребуется посетить этот сейф один раз, а для мультиподписи без скрипта – возможно, два или три раза.

Другой недостаток заключается в том, что при реализации пороговой (threshold) подписи (с заданным количеством требуемых подписей) нельзя узнать, кто именно ее подписал. При пороговой подписи со скриптом Алиса, Боб и Кэрол договариваются (например), что любой из двух подписей будет достаточно для расходования средств. Если подписывают Алиса и Боб, это требует включения подписей от каждого из них в цепочку, доказывая любому знающему их ключи, что они подписали, а Кэрол – нет. При пороговой подписи без скрипта подпись Алисы и Боба неотличима от подписи Алисы и Кэрол, или Боба и Кэрол. Это полезно для конфиденциальности, но означает, что даже если Кэрол скажет, что она не подписывала, она не сможет этого доказать, и это может быть вредно для отчетности и контролируемости.

Для многих пользователей и вариантов использования всегда уменьшенный размер и повышенная конфиденциальность мультиподписей перевешивают периодически возникающие проблемы с созданием и аудитом подписей.

Главный корень (Taproot)

Одна из причин, по которой люди предпочитают работать с Биткойном, состоит в возможности заключать контракты с высокой степенью предсказуемости результатов. Юридические контракты, приводимые в исполнение судом, час-

тично зависят от решений участвующих в деле судей и присяжных. Напротив, контракты Биткойна нередко требуют действий от их участников, но в остальном их исполнение обеспечивается тысячами полных узлов, работающих с функционально идентичным кодом. При наличии одного и того же контракта и одинаковых входных данных каждый полный узел всегда будет выдавать один и тот же результат. Любое отклонение будет обозначать неисправность Биткойна. Судьи и присяжные могут быть гораздо более гибкими, чем программное обеспечение, но если эта гибкость не нужна или нежелательна, предсказуемость контрактов Биткойна становится главным преимуществом.

Если все стороны контракта осознают полную предсказуемость его результата, им нет никакой необходимости продолжать его исполнение. Они могли бы просто выполнить обязательства по контракту и затем расторгнуть его. В обществе большинство контрактов расторгаются именно так: если заинтересованные стороны все устраивает, они никогда не передают контракт на рассмотрение судьи или присяжных. В Биткойне это означает, что любой контракт, для урегулирования которого потребуется значительное количество места в блокчейне, должен содержать условие взаимного согласия (mutual satisfaction).

В MAST и с мультиподписями без скриптов условие о взаимном согласии разрабатывается очень просто. Достаточно лишь превратить один из верхних листьев дерева скриптов в мультиподпись без скрипта между всеми заинтересованными сторонами. На рис. 7.7 уже был показан сложный контракт между несколькими сторонами с простым условием о взаимном согласии. Можно еще больше оптимизировать его, перейдя от мультиподписи со скриптом к мультиподписи без скрипта.

Этот способ достаточно эффективен и конфиденциален. Если используется условие взаимного согласия, нужно предоставить только одну ветвь Меркла, и все, что становится известно, – была ли поставлена подпись (она может быть от одного человека или от тысяч разных участников). Но разработчики в 2018 году поняли, что можно добиться большего, если также использовать оплату по контракту (pay to contract).

В предыдущем описании pay to contract в разделе «Платеж Pay to Contract (P2C)» текст соглашения между Алисой и Бобом был зафиксирован с помощью открытого ключа. Вместо этого можно передать программный код контракта с помощью фиксации корня MAST. Настраиваемый (tweak) в этом случае открытый ключ – это обычный открытый ключ Биткойна, а значит, он может требовать подписи от одного человека или от нескольких (или может быть создан особым образом для предотвращения создания подписи для него). Таким образом, контракт можно обеспечить либо одной подписью всех заинтересованных сторон, либо раскрытием нужной ветви MAST. Дерево обязательств, включающее как открытый ключ, так и MAST, показано на рис. 7.10.

Благодаря этому условие взаимного согласия с мультиподписью является чрезвычайно эффективным и конфиденциальным. Даже более конфиденциальным, нежели может показаться, поскольку любая транзакция, созданная одним пользователем с единственной подписью (или мультиподписью, сгенерированной несколькими контролируемыми им кошельками), выглядит в блокчейне идентично сделке с взаимным согласием. В этом случае нет ника-

кой разницы между расходами миллиона пользователей, участвующих в чрезвычайно сложном контракте, или одного пользователя, тратящего свои накопленные биткойны.

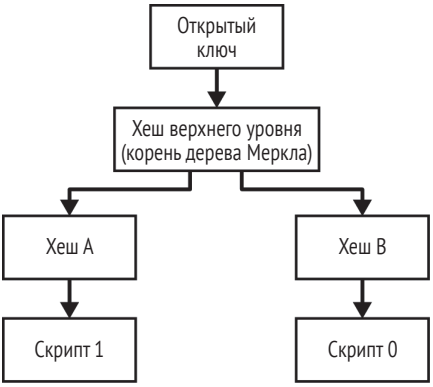


Рис. 7.10 ❖ Главный корень (taproot) с открытым ключом, фиксирующий корень Меркла

Если расходы совершаются только с помощью ключа, например в случае одной подписи или мультиподписи без скриптов, это называют *расходами с ключом* (*keypath spending*). Когда используется дерево скриптов, это называют *расходом по скрипту* (*scriptpath spending*). При расходах с ключом помещенные в цепочку данные представляют собой открытый ключ (в программе свидетеля) и подпись (в стеке свидетеля).

При расходовании по скрипту данные в цепочке также включают открытый ключ, который помещается в программу свидетеля и в данном контексте называется *выходным ключом главного корня* (*taproot output key*). Структура свидетеля включает следующие данные:

- номер версии;
- основной ключ, который существовал до корректировки (твика) корнем Меркла для получения выходного ключа taproot. Этот базовый ключ называется *внутренним ключом taproot* (*taproot internal key*);
- скрипт для выполнения, называемый *скриптом листа* (*leaf script*);
- один 32-байтовый хеш для каждого разветвления дерева Меркла вдоль пути, соединяющего лист с корнем Меркла;
- все необходимые данные для выполнения скрипта (например, подписи или предварительные образы хешей).

Пока известен только один существенный описанный недостаток taproot: контракты, участники которых хотят использовать MAST, но не хотят включать условие о взаимном согласии, должны включать внутренний ключ taproot в блокчейн, что добавляет около 33 дополнительных байтов. Принимая во внимание, что почти все контракты, скорее всего, получат пользу от условия о взаимном согласии или другого условия мультиподписи с применением открытого ключа верхнего уровня, а все пользователи получат выгоду от повышения

анонимности, поскольку их выходные данные будут похожи друг на друга, эти немногочисленные проблемы не были сочтены важными большинством пользователей, принявших участие в активации taproot.

Поддержка taproot была добавлена в Биткойн в ходе софт-форка, активированного в ноябре 2021 года.

Tapscript

Taproot позволяет использовать MAST, но только с немного другой версией языка Bitcoin Script, чем применялась ранее. Новая версия Bitcoin Script называется *tapscript*. Основные отличия таковы:

Изменения в скрипте мультиподписи
: Старые операторы `OP_CHECKMULTISIG` и `OP_CHECKMULTISIGVERIFY` удалены. Они плохо сочетаются с одним из других изменений в софт-форке taproot, а именно с возможностью использования подписей Шнорра (Schnorr signatures) с пакетной проверкой (см. раздел «Подписи Шнорра»). Вместо этого предусмотрен новый оператор `OP_CHECKSIGADD`. При успешной проверке подписи он увеличивает счетчик на единицу, что позволяет удобным образом подсчитать количество успешно проверенных подписей, которое можно сравнить с желаемым количеством успешных подписей для повторной реализации такого же действия, как и в случае `OP_CHECKMULTISIG`.

Изменения во всех подписях
: Все операции с подписями в tapscript используют алгоритм подписи Шнорра, как определено в BIP340. Подробнее о подписях Шнорра будет рассказано в главе 8.

  Кроме того, любая операция проверки подписи, которая, как предполагается, не будет успешной, должна получить значение `OP_FALSE` (также называемое `OP_0`) вместо фактической подписи. Передача чего-либо другого неудавшейся операции проверки подписи приведет к сбою всего скрипта. Это также помогает выполнять пакетную проверку подписей Шнорра.

Операции `OP_SUCCESSx`
: Неиспользуемые в предыдущих версиях Script операции теперь переопределены таким образом, что их применение приводит к успешному выполнению всего скрипта. Это позволяет будущим софт-форкам переопределять их в качестве неуспешных при определенных обстоятельствах. Это является ограничением и поэтому может быть сделано в софт-форке (противоположное определение неуспешной операции как успешной может быть сделано только в хард-форке, что является гораздо более сложным путем обновления).

В этой главе подробно рассмотрены авторизация и аутентификация, но при этом пропущена одна очень важная часть того, как Биткойн производит аутентификацию пользователей, то есть его подписи. Этот вопрос будет рассмотрен в главе 8.

Глава 8

Цифровые подписи

В настоящее время в Биткойне работают два алгоритма подписи: *алгоритм подписи Шнорра* (*Schnorr signature algorithm*) и *алгоритм цифровой подписи на эллиптической кривой* (*Elliptic Curve Digital Signature Algorithm*, *ECDSA*). Они используются для создания цифровых подписей с применением пар секретных/открытых ключей на эллиптической кривой, как описано в разделе «Объяснение криптографии на эллиптических кривых». Они используются для расходования выходов segwit v0 P2WPKH, расходования с ключом segwit v1 P2TR, а также в функциях скриптов `OP_CHECKSIG`, `OP_CHECKSIGVERIFY`, `OP_CHECKMULTISIG`, `OP_CHECKMULTISIGVERIFY` и `OP_CHECKSIGADD`. При выполнении любой из этих команд необходимо предоставить подпись.

Цифровая подпись выполняет в Биткойне три задачи. Во-первых, она доказывает, что распорядитель секретного ключа, который, по сути, является владельцем средств, выдал *авторизацию* на расходование этих средств. Во-вторых, доказательство авторизации является *неопровержимым* (*undeniable*), то есть *безотзывным* (*nonrepudiation*). В-третьих, авторизованная транзакция не может быть изменена неавторизованными (unauthenticated) сторонними лицами – ее *целостность* (*integrity*) неизменна.

Каждый вход транзакции и все подписи, которые он может содержать, полностью независимы от любого другого входа или подписи. Несколько сторон могут совместно создавать транзакции и подписывать только один вход. Некоторые протоколы используют этот факт для создания многосторонних транзакций в целях обеспечения конфиденциальности.

В этой главе подробно рассматривается принцип работы цифровых подписей и способы их использования для доказательства контроля над секретным ключом без раскрытия этого ключа.

Принцип работы цифровых подписей

Цифровая подпись состоит из двух частей. Первая содержит алгоритм создания подписи для сообщения (транзакции) с использованием секретного ключа (ключа подписи). Вторая часть представляет собой алгоритм, с помощью кото-

рого можно проверить подпись при наличии сообщения и соответствующего открытого ключа.

Создание цифровой подписи

В алгоритмах цифровой подписи Биткойна подписываемое «сообщение» – это транзакция, а точнее хеш определенного подмножества данных транзакции, который называют *хеш обязательства* (*commitment hash*, см. раздел «Типы хеша подписи (SIGHASH)»). Ключ подписи – это секретный ключ пользователя. Результатом является подпись:

$$Sig = F_{sig}(F_{hash}(m), x),$$

где

- x – секретный ключ подписи;
- m – сообщение для подписи, хеш обязательства (например, части транзакции);
- F_{hash} – функция хеширования;
- F_{sig} – алгоритм подписи;
- Sig – результирующая подпись.

Более подробно о математике подписей Шнорра и ECDSA можно прочитать в разделах «Подписи Шнорра» и «Подписи ECDSA».

В подписях Шнорра и ECDSA функция F_{sig} формирует подпись Sig, состоящую из двух значений. Между этими двумя значениями разных алгоритмов есть отличия, которые будут рассмотрены позже. После вычисления двух значений они сериализируются в поток байтов. Для подписей ECDSA используется международная стандартная схема кодирования под названием «*Правила отличительного кодирования*» (*Distinguished Encoding Rules*), или сокращенно *DER*. Для подписей Шнорра применяется более простой формат сериализации.

Верификация подписи

Алгоритм верификации подписи получает сообщение (хеш частей транзакции и связанных с ней данных), открытый ключ подписавшего и подпись и возвращает `TRUE` в случае достоверности подписи для этого сообщения и открытого ключа.

Для верификации подписи необходима подпись, сериализованная транзакция, определенные данные о расходуемых средствах и открытый ключ, который соответствует использованному для создания подписи секретному ключу. По сути, верификация подписи подразумевает, что «только обладатель секретного ключа, создавший этот открытый ключ, мог создать эту подпись для этой транзакции».

Типы хеширования подписи (SIGHASH)

Цифровые подписи используются для сообщений, которые в случае с Биткойном сами являются транзакциями. Подпись подтверждает обязательства под-

писавшего лица в отношении конкретных данных транзакции. В простейшей форме подпись применяется почти ко всей транзакции, тем самым подтверждая обязательства по всем входам, выходам и другим полям транзакции. Однако подпись также можно закрепить только за некоторым подмножеством данных в транзакции, что полезно для ряда сценариев, рассматриваемых в этом разделе.

В подписях Биткойна есть способ обозначить часть данных, которая включена в хеш с подписью секретного ключа, с помощью флага SIGHASH. Флаг SIGHASH – это один байт, который добавляется к подписи. Каждая подпись имеет явный или неявный флаг SIGHASH, и этот флаг может отличаться для разных входов. Транзакция с тремя подписанными входами может иметь три подписи с разными флагами SIGHASH, при этом каждая подпись подписывает (фиксирует) разные части транзакции.

Следует помнить, что каждый вход может содержать одну или несколько подписей. В результате на одном входе могут быть подписи с разными флагами SIGHASH, которые фиксируют разные части транзакции. Кроме того, следует учитывать, что транзакции Биткойна могут содержать подписи от разных «владельцев», которые могут подписать только одну подпись в частично созданной транзакции, но при этом взаимодействовать с другими для сбора всех необходимых для совершения корректной транзакции подписей.

Многие типы флагов SIGHASH имеют смысл только при условии, когда несколько участников сотрудничают вне сети Биткойн и обновляют частично подписанную транзакцию.

Существует три типа флагов SIGHASH: ALL, NONE и SINGLE, как показано в табл. 8.1.

Таблица 8.1. Типы флагов SIGHASH и их значения

Флаг SIGHASH	Значение	Описание
ALL	0x01	Подпись применяется ко всем входам и выходам
NONE	0x02	Подпись применяется ко всем входам, ни к одному из выходов
SINGLE	0x03	Подпись применяется ко всем входам, но только к одному выходу с тем же номером индекса, что и подписанный вход

Кроме того, существует флаг-модификатор SIGHASH_ANYONECANPAY, который может сочетаться с любым из перечисленных выше флагов. Когда установлен флаг ANYONECANPAY, подписывается только один вход, оставляя остальные (и их порядковые номера) открытыми для модификации. Флаг ANYONECANPAY имеет значение 0x80 и реализуется побитовым OR (ИЛИ), в результате чего получаются комбинированные флаги, как показано в табл. 8.2.

Таблица 8.2. Типы файлов SIGHASH с модификаторами и их значения

Флаг SIGHASH	Значение	Описание
ALL|ANYONECANPAY	0x81	Подпись применяется к одному входу и всем выходам
NONE|ANYONECANPAY	0x82	Подпись применяется к одному входу, ни к одному из выходов
SINGLE|ANYONECANPAY	0x83	Подпись применяется к одному входу и выходу с одинаковыми номерами индексов

При подписании и проверке флаги SIGHASH применяются так: создается копия транзакции, и определенные поля в ней либо пропускаются, либо сокращаются (устанавливаются в нулевую длину и становятся пустыми). Полученная транзакция сериализуется. В сериализованные данные транзакции включается флаг SIGHASH, и результат хешируется. Полученный хеш-дайджест и есть то самое «сообщение» для подписи. В зависимости от типа используемого флага SIGHASH в транзакцию включаются разные элементы. Включение флага SIGHASH фиксирует и тип SIGHASH, поэтому подпись не может быть изменена (например, майнером).

В разделе «Сериализация подписей ECDSA (DER)» будет показано, что последней частью подписи в кодировке DER будет 01, что является флагом SIGHASH_ALL для подписей ECDSA. Этот флаг блокирует данные транзакции, поэтому подпись Алисы фиксирует состояние всех входов и выходов. Это наиболее распространенная форма подписи.

Перечислим другие типы флагов SIGHASH и рассмотрим варианты их практического использования:

ALL|ANYONECANPAY
: Эта конструкция может использоваться для совершения транзакций в стиле «краудфандинг» (crowdfunding). Желающие собрать средства могут создать транзакцию с одним выходом. На этом выходе производится выплата «целевой» суммы организатору сбора средств. Такая транзакция, разумеется, недействительна, поскольку у нее нет входов. Однако в нее можно внести изменения, добавив свой собственный вход в качестве пожертвования. Для этого нужно подписать свой вклад флагом ALL|ANYONECANPAY. Если количество пожертвований не достигнет размера выходной суммы, транзакция будет недействительной. Каждое пожертвование – это «залог» (pledge), который не может быть получен организатором сбора средств до достижения всей намеченной суммы. К сожалению, этот протокол легко обойти: если организатор сбора средств добавит свой собственный вклад (или вклад того, кто одолжит ему средства), он сможет собирать пожертвования даже без необходимости достижения намеченной целевой суммы.

NONE
: Эта конструкция может использоваться для создания «чека на предъявителя» (bearer check) или «пустого чека» (blank check) на заданную сумму. Она фиксирует все входные данные, но при этом допускает изменение выходных данных. Любой желающий может записать в скрипт выхода свой собственный биткойн-адрес. Это позволяет любому майнеру изменить назначение вывода и присвоить себе средства, но если другие требуемые подписи в транзакции используют SIGHASH_ALL или иной тип флага с фиксацией выхода, это позволяет этим плательщикам изменить назначение, не разрешая сторонним лицам (например, майнерам) изменять выходы.

NONE|ANYONECANPAY
: Эту конструкцию можно использовать для создания «собирателя пыли» (dust collector). Пользователи с крошечными UTXO в кошельках не могут потратить их без расходов, превышающих стоимость UTXO; см. раздел «Не-

выгодные выходы и запрещенная пыль». С помощью этого типа подписи нерентабельные UTXO можно пожертвовать, чтобы все желающие могли собрать их и потратить по своему усмотрению.

Существует несколько предложений по изменению или расширению системы флагов SIGHASH. Наиболее широко обсуждаемым на данный момент является проект BIP118, в котором предлагается добавить два новых флага SIGHASH. Подпись, использующая SIGHASH_ANYPREVOUT, не будет фиксироваться в поле outpoint входа, что позволит применять ее для использования любого предыдущего выхода для конкретной программы свидетеля. Например, если Алиса получает два выхода на одну и ту же сумму для одной и той же программы свидетеля (например, требующей одной подписи из ее кошелька), подпись SIGHASH_ANYPREVOUT для использования одного из этих выходов может быть скопирована и использована для расхода другого выхода по тому же назначению.

Подпись с флагом SIGHASH_ANYPREVOUTANYSCRIPT не фиксирует ни точку выхода, ни сумму, ни программу свидетеля, ни конкретный лист в корневом (taproot) дереве Меркла (дереве скриптов – script tree), что позволяет ей расходовать любой предыдущий выход, обеспечиваемый этой подписью. Например, если Алиса получила два выхода на разные суммы и разные программы свидетелей (например, в одном случае нужна одна подпись, а в другом – ее подпись плюс дополнительные данные), подпись SIGHASH_ANYPREVOUTANYSCRIPT для использования одного из этих выходов может быть скопирована и использована для расхода другого выхода по тому же назначению (при условии что дополнительные данные для второго выхода известны).

Основная целевая область применения двух операций SIGHASH_ANYPREVOUT связана с улучшением платежных каналов, например используемых в сети Lightning Network (LN), хотя уже описаны и другие варианты применения.

Флаги SIGHASH не так уж часто встречаются в качестве опции приложения пользовательского кошелька. Простые приложения для кошельков подписываются флагами SIGHASH_ALL. Более совершенные приложения, такие как LN-узлы, могут использовать альтернативные флаги SIGHASH, однако в них применяются такие протоколы, которые прошли тщательную проверку на предмет влияния альтернативных флагов.

Подписи Шнорра

В 1989 году Клаус Шнорр (Claus Schnorr) опубликовал описание алгоритма подписи, который впоследствии получил его имя. Этот алгоритм не имеет отношения к криптографии на эллиптических кривых (elliptic curve cryptography, ECC), которая используется в Биткойне и других приложениях, однако на сегодняшний день он, пожалуй, больше всего ассоциируется с ECC. Подписи Шнорра характеризуются рядом важных преимуществ:

Доказуемая безопасность
: Математическое доказательство безопасности подписей Шнорра зависит только от сложности решения задачи дискретного логарифма (Discrete Logarithm Problem, DLP), особенно на эллиптических кривых (elliptic curves, EC)

для Биткойна, и способности хеш-функции (например, функции SHA256, применяемой в Биткойне) выдавать непредсказуемые значения. Это так называемая модель случайного оракула (random oracle model, ROM). Другие алгоритмы подписи характеризуются дополнительными особенностями или требуют гораздо больших открытых ключей либо подписей для достижения безопасности на уровне ECC–Шнорра (в случае с угрозами для классических компьютеров; в случае с квантовыми компьютерами другие алгоритмы могут обеспечивать более эффективную защиту).

Линейность

Подписи Шнорра обладают свойством, которое математики называют *линейностью* (*linearity*) и которое применимо к функциям с двумя специфическими свойствами. Первое свойство заключается в том, что при сложении двух или более переменных и последующем выполнении функции по этой сумме получается то же значение, что и при выполнении функции по каждой из переменных независимо и последующем суммировании результатов. Например, $f(x + y + z) == f(x) + f(y) + f(z)$. Это свойство называется *аддитивностью* (additivity), или *суммируемостью*. Второе свойство заключается в том, что умножение переменной и последующее выполнение функции с этим произведением даст то же значение, что и выполнение функции с переменной и последующее умножение на ту же величину, например $f(a \times x) == a \times f(x)$. Это свойство называется *однородностью степени 1* (*homogeneity of degree 1*).

В криптографических операциях некоторые функции могут быть секретными (например, функции с использованием секретных ключей или секретных одноразовых случайных чисел (нонсов – nonces), поэтому возможность получить один и тот же результат при выполнении операции внутри или вне функции облегчает координацию и взаимодействие нескольких сторон без разглашения их секретов. Некоторые преимущества линейности в подписях Шнорра мы рассмотрим в разделах «Мультиподписи без скриптов на основе алгоритма Шнорра» и «Пороговые подписи без скриптов на основе алгоритма Шнорра».

Пакетная верификация

При применении определенным образом (что и происходит в Биткойне) одним из следствий линейности алгоритма Шнорра является возможность одновременной верификации более чем одной подписи Шнорра за меньшее время, чем потребовалось бы для проверки каждой из них по отдельности. Чем больше подписей проверяется за один цикл, тем выше скорость. Для типичного количества подписей в блоке можно провести их пакетную верификацию примерно за половину времени, которое потребовалось бы для проверки каждой из них по отдельности.

Позже в этой главе будет рассмотрен алгоритм подписи Шнорра в том виде, в котором он используется в Биткойне, но для начала есть смысл остановиться на его упрощенной версии и поэтапно перейти к реальному протоколу.

Алиса использует большое случайное число (x), которое называют ее секретным ключом. Она также знает открытую точку на эллиптической кривой

Биткойна, называемую генератором (Generator – *G*, см. раздел «Открытые ключи»). Алиса использует функцию EC-мультипликации для умножения *G* на свой секретный ключ *x*. В этом случае Ш называют *скалярной величиной* (*scalar*), поскольку она фактически масштабирует *G*. В итоге получается значение *xG*, которое называют *открытым ключом* Алисы. Алиса передает свой открытый ключ Бобу. Даже если Боб тоже знает *G*, метод решения DLP не допускает, чтобы Боб мог разделить *xG* на *G* для получения секретного ключа Алисы.

Позже Боб просит Алису идентифицировать себя, подтвердив, что она знает скаляр *x* для полученного ранее Бобом открытого ключа (*xG*). Алиса не может напрямую передать Бобу *x*, потому что это позволило бы ему идентифицировать себя для других людей как ее, поэтому ей нужно доказать свое знание *x*, не раскрывая *x* Бобу. Это и называется *доказательством с нулевым разглашением* (*zero-knowledge proof*). Для этого и служит процесс идентификации Шнорра:

1. Алиса выбирает другое большое случайное число (*k*), которое будем называть *секретным одноразовым случайным числом*, или *секретным нонсом* (*private nonce*). Она вновь использует его в качестве скалярного числа, умножая на *G* для получения *kG*, которое будем называть *открытым одноразовым случайным числом*, или *открытым нонсом* (*public nonce*). Она передает открытый нонс Бобу.
2. Боб выбирает собственное большое случайное число *e*, которое будем называть *скалярным числом запроса* (*challenge scalar*). «Запросом» его называют потому, что Алиса с его помощью должна доказать наличие секретного ключа (*x*) для открытого ключа (*xG*), который она ранее передала Бобу, а «скалярным» – потому, что впоследствии оно будет использоваться для умножения точки на эллиптической кривой.
3. Теперь у Алисы есть числа (скаляры) *x*, *k* и *e*. Она объединяет их для получения конечного скаляра *s* по формуле $s = k + ex$. Она передает *s* Бобу.
4. Теперь Боб знает скаляры *s* и *e*, но не *x* и *k*. Однако Боб знает *xG* и *kG*, и он может вычислить для себя *sG* и *exG*. Это означает, что он может проверить равенство масштабированной версии операции, которую выполнила Алиса: $sG == kG + exG$. Если это равенство равно, то Боб может быть уверен, что Алиса знала *x*, когда генерировала *s*.

Протокол идентификации Шнорра с числами вместо точек на кривой

Интерактивный протокол идентификации Шнорра, возможно, легче понять, если представить себе небезопасное упрощение с заменой каждого из предыдущих значений (включая *G*) простыми целыми числами вместо точек на эллиптической кривой. В качестве примера можно взять простые числа, начиная с 3.

Условия: Алиса выбирает $x = 3$ в качестве секретного ключа. Она умножает его на генератор $G = 5$, чтобы получить свой открытый ключ $xG = 15$. Она передает Бобу 15.

1. Алиса выбирает закрытый нонс $k = 7$ и генерирует открытый нонс $kG = 35$. Она передает Бобу 35.
2. Боб выбирает $e = 11$ и отдает его Алисе.

3. Алиса генерирует $s = 40 = 7 + 11 \times 3$. Она отдает Бобу 40.
4. Боб выводит $sG = 200 = 40 \times 5$ и $exG = 165 = 11 \times 15$. Затем он проверяет, что $200 == 35 + 165$. Обратите внимание, что это та же операция, что и у Алисы, но все значения были увеличены на 5 (значение G).

Разумеется, это слишком упрощенный пример. При работе с простыми целыми числами для получения основного скаляра можно делить произведения на генератор G, но такой способ небезопасен. Именно поэтому важнейшим свойством криптографии на эллиптических кривых, используемой в Биткойне, является простота умножения, но невозможность деления на точку кривой. Кроме того, для таких малых чисел легко найти базовые значения (или допустимые заменители) с помощью перебора. Числа, используемые в Биткойне, значительно больше.

Рассмотрим некоторые особенности интерактивного протокола идентификации Шнорра, которые обеспечивают его безопасность:

Нонс (k)
: На шаге 1 Алиса выбирает число, которое Боб не знает и не может угадать, и сообщает ему масштабированную форму этого числа, kG. В этот момент у Боба также уже имеется ее открытый ключ (xG), который является масштабированной формой ее секретного ключа, x. Это означает, что когда Боб занимается составлением окончательного уравнения ($sG = kG + exG$), есть две независимые переменные, которые Бобу неизвестны (x и k). С помощью элементарной алгебры можно решить уравнение с одной неизвестной переменной, но не с двумя независимыми неизвестными переменными, поэтому наличие нонса Алисы не позволяет Бобу получить ее секретный ключ. Важно отметить, что эта защита зависит от отсутствия возможности угадывания нонсов. Если в нонсе Алисы есть что-то предсказуемое, Боб сможет воспользоваться этим для получения секретного ключа Алисы. Подробнее см. в разделе «Важность случайности в подписях».

Скаляр запроса (e)
: Боб дожидается получения открытого нонса Алисы, а затем на шаге 2 сообщает ей число (скаляр запроса), которое она до этого не знала и не могла угадать. Очень важно, чтобы Боб передал ей скаляр запроса только после подтверждения ее открытого нонса. Представьте, что может произойти, если за Алису захочет выдать себя кто-то, не знающий x, а Боб случайно передаст ему скаляр запроса e до передачи открытого нонса kG. Это позволит злоумышленнику изменить параметры с обеих сторон в проверочном уравнении Боба, $sG == kG + exG$. В частности, он может изменить и sG, и kG. Представьте упрощенную форму этого выражения: $x = y + a$. Если можно изменить и x, и y, то можно отменить a с помощью $x' = (x - a) + a$. Любое выбранное значение x теперь будет отвечать условию этого уравнения. Для реального уравнения злоумышленник просто выбирает случайное число s, генерирует sG, а затем с помощью вычитания на эллиптической кривой выбирает kG, равное $kG = sG - exG$. Он передает Бобу вычисленное значение kG, а потом случайное sG, и Боб принимает его за правильное, поскольку

sG == (*sG* – *exG*) + *exG*. Этим объясняется важность соблюдения очередности операций в протоколе: Боб должен передать Алисе скаляр запроса только после принятия Алисой обязательств в отношении ее открытого нонса.

Описанный здесь интерактивный протокол идентификации частично совпадает с оригинальным описанием Клауса Шнорра, но в нем отсутствуют две важные особенности, необходимые для децентрализованной сети Биткойн. Первая из них заключается в предположении, что Боб будет ждать, пока Алиса подтвердит свой открытый нонс, а затем Боб передаст ей случайный скаляр запроса. В Биткойне отправитель каждой транзакции должен пройти аутентификацию на тысячах полных узлов Биткойна, включая будущие узлы, которые еще не были запущены, но операторы которых однажды захотят убедиться, что полученные ими биткойны были переданы по цепочке, где каждая транзакция была действительной. Любой узел Биткойна, который не может связаться с Алисой, сегодня или в будущем, не сможет подтвердить подлинность ее транзакции и окажется в разногласиях с каждым другим узлом, подтвердившим ее подлинность. Это неприемлемо для такой системы консенсуса, как Биткойн. Для работы Биткойна необходим протокол, который не требует взаимодействия между Алисой и каждым узлом, который хочет ее аутентифицировать.

Простая техника, известная в честь ее первооткрывателей как преобразование Фиата–Шамира (Fiat-Shamir transform), позволяет превратить интерактивный протокол идентификации Шнорра в неинтерактивную схему цифровой подписи. Следует вспомнить о важности шагов 1 и 2, в том числе о необходимости их последовательного выполнения. Алиса должна принять на себя обязательства по непредсказуемому нонсу; Боб должен передать Алисе непредсказуемый скаляр запроса только после получения ее обязательства. Также следует помнить о свойствах безопасных криптографических хеш-функций, которые использовались в других частях этой книги: при одинаковых входных данных они всегда выдают одно и то же значение, но при разных входных данных дают значение, неотличимое от случайных данных.

Это позволяет Алисе выбрать свой секретный нонс, вывести свой открытый нонс, а затем хешировать открытый нонс для получения скаляра запроса. Поскольку Алиса не может предсказать выход хеш-функции (запрос) и поскольку он всегда один и тот же для одного и того же входа (нонса), это гарантированно выдает Алисе случайный запрос, даже если она выбирает нонс и хеширует его самостоятельно. Участие Боба больше не требуется. Она может просто опубликовать свой открытый нонс *kG* и скаляр *s*, и каждый из тысяч полных узлов (прошлых и будущих) может хешировать *kG* для получения *e*, использовать его для получения *exG*, а затем верифицировать *sG* == *kG* + *exG*. Если записать уравнение верификации в явном виде, то получится *sG* == *kG* + *hash*(*kG*) × *xG*.

Для завершения преобразования интерактивного протокола идентификации Шнорра в протокол цифровой подписи, пригодный для Биткойна, необходимо сделать кое-что еще. Нужно, чтобы Алиса не просто доказывала владение своим секретным ключом; необходимо также дать ей возможность зафиксировать сообщение. В частности, чтобы она могла подтвердить данные по отправляемой ею биткойн-транзакции. Благодаря преобразованию Фиата–Шамира обязательство уже создано, так что теперь можно просто обеспечить допол-

нительное обязательство для сообщения. Вместо *hash*(*kG*) теперь можно также зафиксировать сообщение *m* с помощью *hash*(*kG* || *m*), где || означает конкатенацию (объединение).

Итак, версия протокола подписи Шнорра определена, но необходимо решить еще одну проблему, характерную для Биткойна. При деривации ключа BIP32, как описано в разделе «Деривация дочернего открытого ключа», алгоритм неусиленной (unhardened) деривации берет открытый ключ и добавляет к нему несекретное значение для получения производного открытого ключа. Это означает, что к достоверной подписи для одного ключа можно также добавить это несекретное значение и получить подпись для связанного ключа. Эта связанная подпись действительна, но она не была авторизована владельцем секретного ключа, что является серьезным нарушением безопасности. В целях защиты неусиленной деривации BIP32, а также для поддержки нескольких протоколов, которые хотелось бы построить на основе подписей Шнорра, в версии подписей Шнорра для Биткойна, называемой «*подписи Шнорра BIP340 для secp256k1*» (*BIP340 schnorr signatures for secp256k1*), вместе с открытым нонсом и сообщением также фиксируется используемый открытый ключ. Таким образом, полное обязательство имеет вид *hash*(*kG*||*xG*||*m*).

Теперь, когда каждая часть алгоритма подписи Шнорра в BIP340 описана, а также объяснена суть его работы, можно дать определение протоколу. Умножение целых чисел выполняется по *модулю p*, то есть результат операции делится на число *p* (как определено в стандарте secp256k1) и используется остаток. Число *p* очень велико, но если бы оно было равно 3, а результат операции был бы равен 5, то фактически использовалось бы число 2 (т. е. 5, деленное на 3, имеет остаток 2).

Условия: Алиса выбирает большое случайное число (*x*) в качестве своего секретного ключа (либо вручную, либо с помощью протокола типа BIP32 для детерминированной генерации секретного ключа из большого случайного начального значения). Она применяет определенные в secp256k1 параметры (см. «Объяснение криптографии на эллиптических кривых») для умножения генератора *G* на ее скаляр *x*, что в результате дает *xG* (ее открытый ключ). Она передает свой открытый ключ всем, кто в дальнейшем будет проверять подлинность ее транзакций в Биткойне (например, путем включения *xG* в выходные данные транзакции). При готовности к расходованию она генерирует свою подпись:

1. Алиса выбирает большой случайный секретный нонс *k* и определяет открытый нонс *kG*.
2. Алиса выбирает свое сообщение *m* (например, данные транзакции) и генерирует скаляр вызова *e* = *hash*(*kG*||*xG*||*m*).
3. Она генерирует скаляр *s* = *k* + *ex*. Оба этих значения, *kG* и *s*, являются ее подписью. Она передает эту подпись всем, кто хочет ее проверить. Ей также нужно убедиться, что все получат ее сообщение *m*. В Биткойне для этого нужно включить ее подпись в структуру свидетелей ее транзакции, а затем передать эту транзакцию всем узлам.
4. Проверяющие (например, полные узлы) используют *s* для получения *sG* и затем удостоверяются, что *sG* == *kG* + *hash*(*kG*||*xG*||*m*) × *xG*. Если уравне-

ние верно, Алиса подтвердила владение своим секретным ключом *x* (не раскрывая его) и свое обязательство передать сообщение *m* (содержащее данные транзакции).

Сериализация подписей Шнорра

Подпись Шнорра состоит из двух значений, *kG* и *s*. Параметр *kG* – это точка на эллиптической кривой Биткойна (так называемой secp256k1), которая обычно представлена двумя 32-байтными координатами, например (*x*, *y*). Однако здесь нужна только координата *x*, поэтому включено лишь это значение. Если в подписи Шнорра для Биткойна встречается *kG*, помните, что это лишь координата *x* этой точки.

Значение *s* – это скаляр (число для умножения на другие числа). Для кривой secp256k1 Биткойна его длина не может превышать 32 байта.

Хотя и *kG*, и *s* иногда могут иметь меньшее количество байтов, чем 32, такое маловероятно, поэтому они сериализуются как два 32-байтных значения (то есть значения меньше 32 байт содержат нули в старших разрядах). Сериализация происходит в последовательности начиная с *kG* и заканчивая *s*, в результате чего получается ровно 64 байта.

Софт-форк главного корня (taproot), называемый также v1 segwit, привнес в Биткойн подписи Шнорра и на момент написания этой книги был единственным способом их использования в его системе. При использовании taproot для расходования с ключом (keypath) или по скрипту (scriptpath) 64-байтовая подпись Шнорра по умолчанию использует хеш подписи (sighash) типа `SIGHASH_ALL`. Если используется альтернативный sighash или если пользователь не хочет расходовать место для явного указания `SIGHASH_ALL`, к подписи добавляется один дополнительный байт, который определяет хеш подписи. В результате подпись становится 65-байтовой.

Как будет показано далее, использование 64 или 65 байт значительно эффективнее, чем сериализация с помощью подписей ECDSA, описанных в разделе «Сериализация подписей ECDSA (DER)».

Мультиподписи без скриптов на основе алгоритма Шнорра

В протоколе Шнорра с одной подписью, описанном в разделе «Подписи Шнорра», Алиса использует подпись (*kG*, *s*) для общедоступного доказательства владения своим секретным ключом, который в этом случае обозначим как *y*. Представим, что у Боба тоже есть секретный ключ (*z*). Он готов работать с Алисой, чтобы доказать, что они вместе знают $x = y + z$, и при этом ни один из них не раскрывает свой секретный ключ друг другу или кому-либо еще. Рассмотрим протокол подписи Шнорра BIP340 еще раз.

Простой протокол, описанный ниже, не является безопасным по причинам, которые будут описаны далее. Он используется только для демонстрации механизма мультиподписей Шнорра перед описанием соответствующих протоколов, которые считаются безопасными.

Алисе и Бобу нужно определить открытый ключ для *x*, то есть *xG*. Поскольку операции на эллиптических кривых можно использовать для сложения двух EC-точек вместе, они начинают с того, что Алиса определяет *yG*, а Боб – *zG*. Затем они складывают их вместе для создания *xG* = *yG* + *zG*. Точка *xG* – это их *агрегированный открытый ключ* (*aggregated public key*). Для создания подписи они запускают простой протокол мультиподписи:

1. Каждый из них по отдельности выбирает большой случайный секретный нонс, *a* для Алисы и *b* для Боба. Они также отдельно вычисляют соответствующие открытые нонсы *aG* и *bG*. Вместе они получают агрегированный открытый нонс *kG* = *aG* + *bG*.
2. Они договариваются о сообщении для подписи, *m* (например, транзакция), и каждый генерирует свою копию скаляра запроса: *e* = *hash*(*kG*||*xG*||*m*).
3. Алиса создает скаляр *q* = *a* + *ey*. Боб создает скаляр *r* = *b* + *ez*. Они складывают скаляры вместе для получения *s* = *q* + *r*. Их подпись – это два значения, *kG* и *s*.
4. Верификаторы проверяют их открытый ключ и подпись с помощью обычного уравнения: *sG* == *kG* + *hash*(*kG*||*xG*||*m*) × *xG*.

Алиса и Боб доказали, что они знают сумму своих секретных ключей, при этом ни один из них не раскрыл свой секретный ключ другому или кому-либо еще. Этот протокол можно распространить на любое количество участников (например, миллион человек могут доказать, что знают сумму миллиона своих разных ключей).

У этого протокола есть ряд проблем с безопасностью. Наиболее заметная из них заключается в возможности одной из сторон узнать открытые ключи других сторон до передачи своего собственного открытого ключа. Например, Алиса честно генерирует свой открытый ключ *yG* и делится им с Бобом. Боб генерирует свой открытый ключ, *zG* – *yG*. При объединении двух ключей (*yG* + *zG* – *yG*) положительные и отрицательные члены *yG* аннулируются, поэтому открытый ключ представляет собой только секретный ключ для *z* (то есть секретный ключ Боба). Теперь Боб может создать действительную подпись без какой-либо помощи со стороны Алисы. Это называется *атакой с отменой ключа* (*key cancellation attack*).

Существуют различные способы решения проблемы атаки с отменой ключа. Самая простая схема требует от каждого участника зафиксировать свою часть открытого ключа, прежде чем делиться информацией об этом ключе со всеми остальными участниками. Например, Алиса и Боб по отдельности хешируют свои открытые ключи и делятся друг с другом их дайджестами. Получив дайджест друг друга, они могут поделиться своими ключами. Они по отдельности сверяют хеш ключа с ранее предоставленным дайджестом и затем продолжают работу по протоколу в обычном режиме. Это не позволит ни одному из них выбрать открытый ключ, который аннулирует ключи других участников. Однако такую схему можно запросто реализовать неправильно, например наивно использовать ее с незащищенным получением открытого ключа BIP32. Кроме того, она добавляет лишний шаг для взаимодействия участников, что во мно-

гих случаях может быть нежелательно. Для устранения этих недостатков уже предложены более сложные схемы.

Помимо атаки с отменой ключа, также существуют потенциальные варианты атак на нонсы. Напомним, что назначение секретного нонса в том, чтобы никто не смог использовать знания о других значениях в уравнении верификации подписи для вычисления чужого секретного ключа и определения его значения. Чтобы эффективно решить эту задачу, нужно использовать разные нонсы при подписании разных сообщений или изменении других параметров подписи. Различные нонсы не должны быть связаны между собой. В случае мультиподписи каждый участник должен следовать этим правилам, иначе можно поставить под угрозу безопасность других участников. Помимо этого, необходимо предотвращать отмену и другие атаки. Различные протоколы для достижения этих целей идут на определенные компромиссы, поэтому не существует единого протокола мультиподписи, который можно было бы рекомендовать для всех случаев. Вместо этого отметим три протокола из семейства MuSig:

MuSig
: Этот протокол, также называемый *MuSig1*, предполагает три раунда обмена данными в процессе подписания, что аналогично рассмотренному выше процессу. Главным преимуществом MuSig1 является его простота.

MuSig2
: Этот протокол требует только двух раундов обмена данными и в некоторых случаях позволяет совместить один из раундов с обменом ключами. Это способно значительно ускорить подписание для некоторых протоколов, например как это планируется сделать в LN при использовании мультиподписей без скриптов. MuSig2 описан в BIP327 (единственный протокол мультиподписи без скриптов, который на момент написания этой книги имеет BIP).

MuSig-DN
: DN означает «*детерминированный нонс*» (*Deterministic Nonce*), который позволяет избежать проблемы, известной как *атака на повторную сессию* (*repeated session attack*). Его нельзя сочетать с обменом ключами, и он значительно сложнее в реализации, чем MuSig или MuSig2.

Для большинства приложений MuSig2 является лучшим протоколом мультиподписи, доступным на момент написания статьи.

Пороговые подписи без скриптов на основе алгоритма Шнорра

Протоколы мультиподписи без скриптов работают только для подписи *k*-из-*k*. Каждый участник с частичным открытым ключом, который становится частью объединенного открытого ключа, должен предоставить частичную подпись и частичный нонс для окончательной подписи. Впрочем, в некоторых случаях участники предпочитают разрешить подписываться подмножеству из их числа, например *t*-из-*k*, где пороговое (threshold – *t*) число участников может подписывать ключ, созданный *k* участниками. Такой тип называется «*пороговой подписью*» (*threshold signature*).

О пороговых подписях на основе скриптов рассказано в разделе «Мультиподписи со скриптами». Точно так же, как мультиподписи без скриптов экономят место и повышают конфиденциальность по сравнению с мультиподписями со скриптами, *пороговые подписи без скриптов* (*scriptless threshold signatures*) экономят место и повышают конфиденциальность по сравнению с *пороговыми подписями со скриптами* (*scripted threshold signatures*). Для тех, кто не участвует в подписании, *пороговая подпись без скриптов* выглядит как любая другая подпись, которая могла быть создана пользователем с одной подписью или с помощью протокола мультиподписей без скриптов.

Известны различные методы создания пороговых подписей без скриптов, и самый простой из них – слегка модифицированный способ создания мультиподписей без скриптов, описанный ранее. Этот протокол также зависит от верифицируемого обмена секретами (который сам зависит от безопасного обмена секретами).

Базовый обмен секретами может работать через простое разделение. У Алисы есть секретное число, которое она делит на три части равной длины и передает Бобу, Кэрол и Дэну. Эти трое могут объединить полученные ими части, которые также называются *долями* (*shares*), в правильном порядке, чтобы восстановить секрет Алисы. Более сложная схема предполагает добавление Алисой к каждой доле некоторой дополнительной информации, называемой корректирующим кодом (correction code), что позволяет восстановить число любым двум из них. Эта схема небезопасна, поскольку каждая доля дает ее владельцу частичное знание секрета Алисы и способствует более легкому угадыванию секрета Алисы участником, чем тем, кто не обладает долей.

Безопасная схема разделения секрета предотвращает получение участниками информации о нем до тех пор, пока они не объединят минимальное пороговое количество долей. Например, Алиса может выбрать пороговое значение 2, если она хочет, чтобы два любых участника, Боб, Кэрол или Дэн, смогли восстановить ее секрет. Самым известным алгоритмом безопасного разделения секретов является *схема разделения секретов Шамира* (*Shamir's Secret Sharing Scheme*), которую обычно сокращенно называют SSSS и которая названа в честь ее первооткрывателя, одного из тех самых авторов преобразования Фиата–Шамира, которое было описано в разделе «Подписи Шнорра».

В некоторых рассматриваемых здесь криптографических протоколах, таких как схемы пороговых подписей без скриптов, Бобу, Кэрол и Дэну очень важно быть уверенными в том, что Алиса следовала своей части протокола корректно. Они должны знать, что все созданные ею доли получены из одного и того же секрета, что Алиса использовала заявленное ею пороговое значение и что она выдала каждому из них разные доли. Протокол, который способен выполнить все это и оставаться при этом безопасной схемой разделения секретов, называют *верифицируемой схемой распределения секретов* (*verifiable secret sharing scheme*).

Чтобы понять, как мультиподписи и верифицируемый обмен секретами используются Алисой, Бобом и Кэрол, предположим, что каждый из них хочет получить средства, которые могут быть потрачены двумя любыми из них. Они сотрудничают, как описано в разделе «Мультиподписи без сценариев на основе Шнорра», чтобы создать обычный открытый ключ мультиподписи для при-

нятия средств (*k*-из-*k*). Затем каждый из них извлекает две секретные доли из своего секретного ключа – по одной для каждого из двух других участников. Эти доли позволяют любому из них восстановить частичный секретный ключ для мультиподписи. Каждый участник передает одну из своих секретных долей двум другим участникам, в результате чего каждый участник хранит свой собственный частичный секретный ключ и по одной доле от каждого другого. Далее каждый участник верифицирует аутентичность и уникальность полученных им долей, сравнивая их с долями от других участников.

Позже, когда (например) Алиса и Боб захотят создать пороговую подпись без скриптов, не привлекая Кэрол, они обменяются с Кэрол двумя имеющимися у них долями. Это позволяет им восстановить частичный секретный ключ Кэрол. Алиса и Боб также имеют свои секретные ключи, позволяющие им создать мультиподпись без скрипта со всеми тремя необходимыми ключами.

Иными словами, описанная схема пороговой подписи без скрипта – это то же самое, что и схема мультиподписи без скрипта, за исключением того, что у кворума (порогового количества) участников имеется возможность восстановить частичные секретные ключи всех остальных участников, которые не могут или не хотят ставить подпись.

Все это указывает на несколько моментов, которые следует учитывать при выборе протокола пороговой подписи без скрипта:

Отсутствие подотчетности
: Поскольку Алиса и Боб восстанавливают частичный секретный ключ Кэрол, не может быть принципиальной разницы между мультиподписью без скрипта, созданной процессом с участием Кэрол и без нее. Даже если Алиса, Боб или Кэрол заявят, что они не ставили подпись, у них не будет гарантированного способа доказать, что они не помогали в создании подписи. Если необходимо точно знать, кто из участников группы поставил подпись, придется использовать скрипт.

Манипуляционные атаки
: Представим, что Боб сообщает Алисе о том, что Кэрол недоступна, и они вместе работают над восстановлением частичного секретного ключа Кэрол. Затем Боб сообщает Кэрол, что Алиса недоступна, и они вместе восстанавливают частичный секретный ключ Алисы. Теперь у Боба есть его собственный секретный ключ плюс ключи Алисы и Кэрол, что позволяет ему тратить средства самостоятельно, без их участия. С этой атакой можно справиться, если все участники договорятся общаться только по схеме с возможностью просмотра всех сообщений друг друга (например, если Боб сообщает Алисе, что Кэрол недоступна, Кэрол сможет увидеть это сообщение до того, как начнет работать с Бобом). На момент написания книги велись поиски других, возможно более надежных, решений этой проблемы.

Пока ни один протокол пороговой подписи без скриптов не был предложен в качестве BIP, хотя многие участники проекта Биткойн провели значительные исследования на эту тему. Авторы ожидают, что после публикации данной книги появятся решения, прошедшие рецензирование.

Подписи ECDSA

К несчастью для перспектив развития Биткойна и многих других приложений, Клаус Шнорр запатентовал открытый им алгоритм и почти два десятилетия препятствовал его использованию в открытых стандартах и программном обеспечении с открытым исходным кодом. В начале 1990-х годов специалисты по криптографии, лишенные возможности использовать схему подписи Шнорра, разработали альтернативную структуру под названием *алгоритм цифровой подписи* (*Digital Signature Algorithm*, *DSA*), а его адаптация на эллиптические кривые (*elliptic curves*) получила название ECDSA.

Схема ECDSA и стандартизированные параметры для предложенных кривых, с которыми она может использоваться, к моменту начала разработки Биткойна в 2007 году уже имели широкое распространение в криптографических библиотеках. Почти наверняка именно по этой причине ECDSA был единственным протоколом цифровой подписи, который поддерживался в Биткойне с момента выхода его первой версии до активации софт-форка taproot в 2021 году. ECDSA поддерживается и сегодня для всех транзакций, не связанных с taproot. Среди отличий от подписей Шнорра можно выделить следующие:

Большая сложность
: Как будет показано далее, ECDSA требует больше операций для создания или проверки подписи, чем протокол подписи Шнорра. С точки зрения реализации он ненамного сложнее, но эта дополнительная сложность делает ECDSA менее гибким, менее производительным и сложным инструментом доказательства безопасности.

Меньшая доказательная защищенность
: Интерактивный протокол идентификации подписи Шнорра зависит только от стойкости задачи дискретного логарифма на эллиптической кривой (Elliptic Curve Discrete Logarithm Problem, ECDLP). Неинтерактивный протокол аутентификации, используемый в Биткойне, также основан на модели случайного оракула (Random Oracle Model, ROM). Однако дополнительная сложность ECDSA не позволила полностью доказать ее безопасность (насколько известно). И хотя авторы не являются экспертами в доказательстве криптографических алгоритмов, представляется маловероятным, что спустя 30 лет для доказательства ECDSA потребуются только те же два предположения, что и для Шнорра.

Нелинейность
: Подписи ECDSA невозможно просто объединить для создания мультиподписей без скриптов или использовать в смежных расширенных приложениях, таких как многосторонние адаптеры подписей. Существуют обходные пути решения этой проблемы, но они сопряжены с дополнительными сложностями, значительно замедляющими выполнение операций, и в некоторых случаях это приводит к случайной утечке секретных ключей.

Алгоритм ECDSA

Рассмотрим математический принцип ECDSA. Подписи создаются с помощью функции F_{sig}. Она создает подпись, состоящую из двух значений. В ECDSA эти два значения называются R и s.

Алгоритм подписи сначала генерирует секретный нонс (k) и на его основе создает открытый нонс (K). Значение R в цифровой подписи – это координата x нонса K.

Далее алгоритм вычисляет значение s для подписи. Как и в случае с подписями Шнорра, операции с целыми числами выполняются по модулю p:

$$s = k^{-1}(Hash(m) + x \times R),$$

где

- k – секретный нонс;
- R – координата x открытого нонса;
- x – секретный ключ Алисы;
- m – сообщение (данные транзакции).

Верификация – это обратная функция генерации подписи, использующая значения R, s и открытый ключ для вычисления значения K, которое является точкой на эллиптической кривой (открытый нонс, используемый при создании подписи):

$$K = s^{-1} \times Hash(m) \times G + s^{-1} \times R \times X,$$

где

- R и s – значения подписи;
- X – открытый ключ Алисы;
- m – сообщение (данные транзакции, которая была подписана);
- G – точка генератора эллиптической кривой.

Если координата x вычисленной точки K равна R, то проверяющий может сделать вывод, что подпись действительна.

ECDSA – это довольно сложная математика, полное объяснение которой выходит за рамки данной книги. В интернете можно найти множество отличных руководств в помощь тем, кто хочет разобраться в этом пошагово. Для этого можно набрать в поисковике «Объяснение ECDSA» (ECDSA explained).

Сериализация подписей ECDSA (DER)

Рассмотрим следующую DER-кодированную подпись:

```
3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204
b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e381301
```

Эта подпись представляет собой сериализованный поток байтов значений R и s. Она создается подписывающим лицом для подтверждения владения сек-

ретным ключом, разрешенным для расходования выходных данных. Формат сериализации включает девять элементов:

- `0x30`, указывает на начало последовательности DER;
- `0x45`, длина последовательности (69 байт);
- `0x02`, следующее за ней целочисленное значение;
- `0x21`, длина целого числа (33 байта);
- `R`, `00884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb`;
- `0x02`, далее следует другое целое число;
- `0x20`, длина целого числа (32 байта);
- `S`, `4b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813`;
- суффикс (`0x01`), указывающий на тип используемого хеша (`SIGHASH_ALL`).

Важность случайности в подписях

Как было показано в разделах «Подписи Шнорра» и «Подписи ECDSA», алгоритм создания подписи использует случайное число *k* в качестве основы для пары секретного и открытого нонсов. Значение *k* не играет роли, главное, чтобы оно было случайным. Если подписи с одним и тем же секретным ключом используют секретный нонс *k* с разными сообщениями (транзакциями), то секретный ключ подписи может быть вычислен кем угодно. Повторное использование одного и того же значения *k* в алгоритме подписи приводит к раскрытию секретного ключа!

Если одно и то же значение *k* используется в алгоритме подписи в двух разных транзакциях, секретный ключ может быть вычислен и раскрыт для всего света!

Это не только теоретическая возможность. В нескольких различных вариантах реализации алгоритмов подписания транзакций в Биткойне эта проблема приводила к раскрытию секретных ключей. У людей были украдены средства из-за непреднамеренного повторного использования значения *k*. Наиболее распространенной причиной повторного использования значения *k* является неправильно настроенный генератор случайных чисел.

Чтобы избежать этой уязвимости, лучше всего генерировать *k* не с помощью генератора случайных чисел с энтропией, а использовать процесс, который частично состоит из самих данных транзакции и секретного ключа, применяемого для подписи. Это гарантирует, что каждая транзакция будет генерировать свой *k*. Стандартный алгоритм детерминированной инициализации *k* для ECDSA определен в RFC6979 (https://datatracker.ietf.org/doc/html/rfc6979), опубликованном Инженерным советом Интернета (Internet Engineering Task Force). Для подписей Шнорра протокол BIP340 рекомендует алгоритм подписи по умолчанию.

В BIP340 и RFC6979 можно генерировать *k* полностью детерминированно, то есть одни и те же данные транзакции всегда будут генерировать одно и то же значение *k*. Многие кошельки так и поступают, поскольку это упрощает соз-

дание тестов для проверки корректности генерации значений *k* в критически важном для безопасности коде подписи. Если эти данные представляют собой энтропию, то даже при подписании точно таких же данных транзакции будет получено другое значение *k*. Это может повысить защиту от атак с использованием побочных каналов (sidechannel attack) и внедрения ошибок (fault-injection attack).

Если вы используете алгоритм для подписи транзакций в Биткойне, необходимо воспользоваться BIP340, RFC6979 или аналогичным алгоритмом для гарантированной генерации разных *k* для каждой транзакции.

Новый алгоритм подписи Segregated Witness

Подписи в транзакциях Биткойна применяются к *хешу обязательства* (*commitment hash*), который вычисляется из данных транзакции и блокирует определенные части этой информации, указывая на обязательства подписанта в отношении этих значений. Например, в простой подписи типа SIGHASH_ALL хеш обязательства включает все входы и выходы.

К сожалению, способ вычисления хешей обязательств, использовавшийся в старых версиях системы, привел к тому, что проверяющий подпись узел был вынужден выполнять значительное количество хеш-вычислений. В частности, число хеш-операций увеличивается примерно квадратично с ростом числа входов транзакции. Поэтому злоумышленник может создать транзакцию с очень большим количеством операций подписи, в результате чего вся сеть Биткойна будет вынуждена выполнить сотни или тысячи хеш-операций для верификации такой транзакции.

Появление Segwit дало возможность решить эту проблему, изменив способ вычисления хеша обязательства. Для программ свидетелей segwit версии 0 верификация подписи происходит с помощью улучшенного алгоритма хеширования обязательств, как указано в BIP143.

Новый алгоритм дает возможность увеличить количество хеш-операций на величину $O(n)$ по сравнению с количеством операций подписи, что уменьшает возможность создания атак типа «отказ в обслуживании» (denial-of-service) с применением чрезмерно усложненных транзакций.

В этой главе было рассказано о подписях Шнорра и ECDSA для Биткойна. Благодаря им полные узлы обеспечивают аутентификацию транзакций, для того чтобы потратить биткойны, полученные с помощью ключа, мог только владелец этого ключа. Также было рассмотрено несколько вариантов применения подписей, таких как мультиподписи без скриптов и пороговые подписи без скриптов, которые можно использовать для повышения эффективности и конфиденциальности Биткойна. В последних нескольких главах на примере Биткойна было описано, как создавать транзакции, как защитить их с помощью авторизации и аутентификации, а также как их подписывать. Далее речь пойдет о способах поощрения майнеров к заверению транзакций путем добавления комиссии к создаваемым нами транзакциям.

Глава 9

Комиссия за транзакцию

Цифровая подпись, которую Алиса создала в главе 8, доказывает только тот факт, что она знает свой секретный ключ и что она совершила платежную операцию с Бобом. Она может создать другую подпись, которая вместо этого фиксирует транзакцию для выплаты Кэрол – транзакцию, которая тратит тот же самый выход (биткойны), который она использовала для выплаты Бобу. Теперь это *конфликтующие транзакции* (*conflicting transactions*), поскольку в действующий блокчейн с наибольшим количеством доказательств работы и используемый полными узлами для определения контроля над биткойнами может быть включена только одна транзакция с конкретным выходом.

Для защиты от конфликтующих транзакций Бобу было бы разумнее подождать, пока транзакция от Алисы не будет включена в блокчейн на достаточную глубину, чтобы считать полученные деньги своими (см. раздел «Подтверждения»). Чтобы транзакция Алисы была включена в блокчейн, она должна входить в блок транзакций. За определенный промежуток времени создается ограниченное количество блоков, и каждый блок занимает ограниченный объем пространства. Только майнер, создающий этот блок, может выбирать транзакции для включения в него. Майнеры могут проводить отбор по любым критериям, включая полный отказ от включения транзакций.

При использовании термина «транзакции» в этой главе подразумеваются все транзакции в блоке, кроме первой. Первая транзакция в блоке – это *транзакция coinbase*, описанная в разделе «Транзакции Coinbase», которая позволяет майнеру блока получить свое вознаграждение за добычу блока. В отличие от других, транзакция coinbase не расходует результат предыдущей транзакции, а также представляет собой исключение из ряда других правил, применимых к другим транзакциям. Транзакции coinbase не платят комиссию, не нуждаются в повышении комиссии, не подвержены пиннингу и в целом не представляют интереса для последующего обсуждения комиссий, поэтому в этой главе они не рассматриваются.

Практически все майнеры при выборе транзакций для включения в блоки руководствуются критерием получения максимального дохода. Биткойн специально разрабатывался с учетом этого, предоставив механизм для отчисления денег майнеру за включение транзакции в блок. Этот механизм называют

«*комиссия за транзакцию*» (*transaction fee*), несмотря на то что это не комиссия в обычном понимании данного слова. Это не сумма, установленная протоколом или конкретным майнером, – скорее, это напоминает ставку на аукционе. Покупаемый товар – это часть ограниченного пространства в блоке, которую займет транзакция. Майнеры подбирают набор транзакций, чьи ставки позволят им получить наибольший доход.

В этой главе рассматриваются различные аспекты этих ставок – транзакционных сборов, а также их роль в создании и управлении транзакциями Биткойна.

Кто платит комиссию за транзакцию?

Большинство платежных систем взимают определенную комиссию за транзакцию, но зачастую эта комиссия скрыта от обычных покупателей. Например, продавец может рекламировать один и тот же товар по одной и той же цене вне зависимости от способа оплаты – наличными или кредитной картой, хотя его платежный процессор может взимать с него более высокую комиссию за кредитные транзакции, чем его банк за наличные депозиты.

В Биткойне каждый перевод биткойнов должен быть подтвержден (обычно подписью), поэтому транзакция не может содержать комиссию без разрешения отправителя. Получатель транзакции может оплатить комиссию в другой транзакции, и это будет показано далее, но если нужно, чтобы какая-то отдельная транзакция оплачивала собственную комиссию, эта комиссия должна быть чем-то согласованным с отправителем. Она не может быть скрытой.

Биткойн-транзакции устроены таким образом, чтобы в них не требовалось дополнительного места для фиксации оплаты комиссии. Поэтому, несмотря на возможность оплатить комиссию в другой транзакции, наиболее эффективным (и, следовательно, дешевым) способом является оплата комиссии в ходе одной транзакции.

В Биткойне комиссия – это ставка, и от ее размера зависит количество времени, которое займет подтверждение транзакции. В быстром подтверждении платежа заинтересованы, как правило, и плательщики, и получатели. Поэтому обычное разрешение выбирать размер комиссии только плательщикам иногда может быть проблемой. Решение этого вопроса будет рассмотрено в разделе «Повышение комиссии Replace By Fee (RBF)». Однако во многих распространенных платежных системах именно плательщики больше всего заинтересованы в быстром подтверждении транзакции – то есть готовы платить более высокие комиссии.

По этим причинам – как техническим, так и практическим – в Биткойне принято, что комиссию за транзакцию платит сторона, расходующая средства. Существуют исключения, например, для продавцов, принимающих неподтвержденные транзакции, и в протоколах, которые не транслируют транзакции сразу после их подписания (что не позволяет расходующей стороне выбрать подходящую комиссию в соответствии с текущими рыночными условиями). Эти исключения будут рассмотрены позже.

Комиссии и ставки комиссий

За каждую транзакцию вне зависимости от ее размера взимается только одна комиссия. В то же время чем больше становится размер транзакций, тем меньшее их количество майнер сможет вместить в блок. По этой причине майнеры оценивают транзакции так же, как на рынке сравнивают несколько равнозначных товаров: они делят цену на количество.

Если стоимость нескольких мешков риса можно разделить на вес каждого мешка для определения самой низкой цены за вес (самая выгодная сделка), то при оценке транзакций майнеры делят комиссию за них на их размер (также называемый весом – weight) для определения самой высокой комиссии за вес (наибольший доход). В Биткойне для обозначения размера транзакции, деленного на ее вес, используется термин *ставка комиссии* (*fee rate*). В связи с различными переменами в Биткойне за годы его существования размер комиссии мог выражаться в разных единицах:

- BTC/байты (устаревшая единица, редко используемая в наши дни);
- BTC/килобайты (устаревшая, редко используемая единица);
- BTC/Vbytes (1 Vbyte равен 4 единицам веса; используется редко);
- BTC/Kilo-Vbyte (используется в основном в Bitcoin Core);
- сатоши/Vbyte (наиболее распространенная единица в настоящее время);
- сатоши/вес (также широко распространена в настоящее время).

Мы рекомендуем для отображения ставок комиссии использовать единицы сатоши/вбайт (sat/Vbyte) или сатоши/вес (sat/weight).

Будьте осторожны при вводе данных о ставках комиссии. Если пользователь скопирует и вставит ставку комиссии, выведенную в одном знаменателе, в поле с другим знаменателем, он может переплатить в 1000 раз. Если вместо этого они поменяют числитель, то теоретически могут переплатить в 100 000 000 раз. Кошельки должны затруднять пользователю выплату чрезмерной комиссии и, возможно, предлагать подтверждение любой комиссии, которая не была сгенерирована самим кошельком с помощью надежного источника данных.

Завышенной, или *абсурдной*, *комиссией* (*absurd fee*) называют любую ставку, которая значительно превышает ожидаемую в настоящее время сумму для подтверждения транзакции в следующем блоке. Обратите внимание, что кошельки не должны полностью запрещать пользователям выбирать завышенную ставку комиссии. Они должны лишь затруднять ее случайное использование. В редких случаях у пользователей могут быть обоснованные причины для завышения комиссии.

Оценка приемлемых ставок комиссии

Существует мнение, что можно заплатить меньшую ставку комиссии, если вы готовы дольше ждать подтверждения транзакции. За исключением того, что при слишком низкой ставке комиссии можно не дождаться подтверждения транзакции. Поскольку ставки комиссии формируются в ходе открытого аукциона по продаже места в блоках, нельзя точно предсказать размер комиссии,

которую нужно заплатить для подтверждения транзакции к определенному времени. Однако можно сделать приблизительную оценку, основываясь на ставках комиссий по другим недавно проведенным транзакциям.

Полный узел может записывать три части информации о каждой просмотренной транзакции: время (высота блока) первого получения транзакции, высота блока в момент подтверждения транзакции и размер уплаченной комиссии по этой транзакции. Сгруппировав вместе транзакции, которые поступили на одинаковой высоте, были подтверждены на одинаковой высоте и заплатили одинаковые комиссии, можно рассчитать количество блоков, потребовавшихся для подтверждения транзакций с определенной ставкой комиссии. Затем можно считать, что транзакции с аналогичной ставкой комиссии теперь будут подтверждаться через такое же количество блоков. В Bitcoin Core есть оценщик ставки комиссии, основанный на этих принципах. Его можно вызвать с помощью RPC `estimatesmartfee` с параметром, указывающим на количество блоков, которые вы готовы ждать, прежде чем транзакция будет подтверждена с высокой вероятностью (например, 144 блока, то есть примерно 1 день):

```
$ bitcoin-cli -named estimatesmartfee conf_target=144
{
  "feerate": 0.00006570,
  "blocks": 144
}
```

Многие интернет-сервисы также предлагают оценку стоимости услуг в виде API. Актуальный список см. на сайте https://www.lopp.net/bitcoin-information/fee-estimates.html.

Как уже говорилось, оценка ставок комиссии не бывает идеальной. Одна из распространенных проблем заключается в потенциальных колебаниях фундаментального спроса, которые могут изменить равновесие, повышая цены (комиссии) до новых высот либо снижая их до минимума. Если ставки комиссии снижаются, то транзакция, которая ранее оплачивалась по обычной ставке, теперь может быть проведена с высокой комиссией, и она будет подтверждена даже раньше, чем ожидалось. Невозможно снизить ставку комиссии для уже отправленной транзакции, поэтому придется платить по более высокой ставке. Но при их росте возникает необходимость в соответствующих методах повышения ставок по этим транзакциям. Это называется *повышением комиссии* (*fee bumping*). В Биткойне широко используются два типа повышения комиссии: *замена за комиссию* (*replace by fee*, *RBF*) и *дочерняя оплата за родителя* (*child pays for parent*, *CPFP*).

Повышение комиссии Replace By Fee (RBF)

Чтобы увеличить плату за транзакцию с помощью метода повышения комиссии RBF, необходимо создать конфликтующую версию транзакции, которая платит более высокую комиссию. Две или более транзакций считаются конфликтующими, если только одна из них может быть включена в действующий блокчейн. Это вынуждает майнера выбирать только одну из них. Конфликты

возникают, когда две или более транзакции пытаются потратить один и тот же UTXO, то есть каждая из них включает в себя вход с одной и той же точкой выхода (ссылка на выход предыдущей транзакции).

Для предотвращения возможности расходования больших объемов сетевого трафика за счет неограниченного количества конфликтующих транзакций и их отправки через всю сеть ретранслирующих полных узлов в Bitcoin Core и полных узлах, поддерживающих замену транзакций, предусмотрено условие о выплате более высокой комиссии за каждую заменяющую транзакцию по сравнению с заменяемой. В настоящее время Bitcoin Core также требует, чтобы заменяющая транзакция платила более высокую общую комиссию, чем исходная транзакция. Впрочем, это требование влечет за собой ряд нежелательных побочных эффектов, и на момент написания данной книги разработчики уже искали способы их устранения.

В настоящее время Bitcoin Core поддерживает две разновидности RBF:

RBF по согласию (opt-in RBF)
: Неподтвержденная транзакция может сигнализировать майнерам и полным узлам, что ее создатель хочет дать разрешение на ее замену версией с более высокой ставкой комиссии. Этот сигнал и правила его использования определены в BIP125. На момент написания данной книги он был включен по умолчанию в Bitcoin Core на протяжении нескольких лет.

Полный RBF
: Любая неподтвержденная транзакция может быть заменена версией с более высокой комиссией. На данный момент это опционально включено в Bitcoin Core (но по умолчанию отключено).

Почему существует две разновидности RBF?

Причина существования двух разных версий RBF связана с разногласиями по поводу полного RBF. Ранние версии Биткойна позволяли заменять транзакции, но затем это право было отключено на протяжении нескольких релизов. В это время майнер или полный узел с программным обеспечением, которое сейчас называется Bitcoin Core, не имел возможности заменить первую полученную версию неподтвержденной транзакции на любую другую. Некоторые продавцы привыкли к такой практике. Они полагали, что любая действительная неподтвержденная транзакция, оплатившая соответствующую комиссию, в конечном итоге станет подтвержденной, и поэтому предоставляли свои товары или услуги сразу же после получения такой неподтвержденной транзакции.

Однако протокол Биткойна не дает никаких гарантий относительно окончательного подтверждения любой неподтвержденной транзакции. Как говорилось ранее в этой главе, каждый майнер может сам выбирать транзакции для подтверждения, в том числе и версии этих транзакций. Bitcoin Core – это программное обеспечение с открытым исходным кодом, поэтому любой, у кого есть его копия, может добавить (или удалить) возможность замены транзакций. Даже если бы Bitcoin Core не был открытым, система Биткойн – это открытый протокол, который может быть реализован с нуля достаточно компетентным программистом, при этом он может включать или не включать функцию замены транзакций.

Замена транзакций ломает предположение некоторых продавцов о возможности подтверждения каждой приемлемой неподтвержденной транзакции. Альтернативная версия транзакции может оплачивать те же выходы, что и оригинал, но она не обязательно должна оплачивать какой-либо из них. Если первая версия неподтвержденной транзакции платит продавцу, вторая версия может и не платить. Если продавец предоставил товары или услуги на основе первой версии, но вторая версия была подтверждена, то он не получит оплату за свои расходы.

Некоторые продавцы и поддерживающие их пользователи просили не возобновлять замену транзакций в Bitcoin Core. Другие же указали на то, что замена транзакций дает определенные преимущества, в том числе возможность взимать комиссию с транзакций, которые изначально платили слишком низкую ставку комиссии.

В конце концов разработчики Bitcoin Core пошли на компромисс. Вместо разрешения замены каждой неподтвержденной транзакции (полный RBF) они запрограммировали Bitcoin Core на замену только транзакций с сигналом о согласии на замену (opt-in RBF). Продавцы могут проверять получаемые ими транзакции на наличие сигнала opt-in и обрабатывать их иначе, чем транзакции без него.

Это не меняет сути проблемы: любой может изменить свою копию Bitcoin Core или создать новую реализацию с разрешением полного RBF, и некоторые разработчики даже сделали это, но, похоже, мало кто воспользовался их программным обеспечением.

Спустя несколько лет создатели Bitcoin Core немного изменили условия компромисса. В дополнение к сохранению RBF по согласию они добавили опцию включения полного RBF. Если достаточное количество майнинговых мощностей и ретрансляционных узлов включат эту опцию, то любая неподтвержденная транзакция со временем будет заменена версией с более высокой ставкой комиссии. На данный момент пока еще не ясно, произошло это или нет.

Если вы планируете использовать функцию повышения комиссии RBF, сначала следует выбрать кошелек с ее поддержкой, например один из отмеченных в «Sending support» на сайте https://bitcoinops.org/en/compatibility/#replace-by-fee-rbf.

Если вы как разработчик планируете реализовать функцию повышения комиссии RBF, сначала необходимо решить, будет ли это вариант opt-in RBF или полный RBF. На момент написания книги opt-in RBF был единственным надежным решением. Даже если полный RBF станет достаточно эффективным, скорее всего, еще несколько лет замена транзакций opt-in будет подтверждаться немного быстрее, чем полные RBF-замены. Если вы выберете opt-in RBF, в вашем кошельке должна быть реализована передача сигнала согласно BIP125, которая представляет собой простую модификацию любого из полей последовательности в транзакции (см. раздел «Последовательность»). Если вы выберете полный RBF, вам не нужно добавлять в транзакции никаких сигналов. Все остальное, связанное с RBF, одинаково для обоих вариантов.

Когда требуется заменить транзакцию, просто создайте новую транзакцию с расходом по крайней мере одного из тех же UTXO, что и в исходной заменяемой транзакции. Скорее всего, вы захотите сохранить в транзакции те же выходы, которые оплачивает получатель (или получатели). Можно заплатить

повышенную комиссию, уменьшив значение изменяемого выхода или добавив в транзакцию дополнительные входы. Разработчикам следует обеспечить пользователей интерфейсом для увеличения комиссии, который выполняет всю эту работу за них и просто спрашивает (или предлагает) размер увеличения ставки комиссии.

Будьте очень осторожны при создании нескольких замен одной и той же транзакции. Следует убедиться, что все версии транзакций не конфликтуют друг с другом. Если конфликтуют не все, возможно подтверждение нескольких отдельных транзакций, в результате чего получится переплата получателям. Например:

- транзакция версии 0 включает вход A;
- транзакция версии 1 включает входы A и B (например, для оплаты дополнительных комиссий пришлось добавить вход B);
- транзакция версии 2 включает входы B и C (например, для оплаты дополнительных комиссий пришлось добавить вход C, но C оказался достаточно большим, и вход A больше не нужен).

В этом сценарии любой майнер, сохранивший транзакцию версии 0, сможет подтвердить и ее, и транзакцию версии 2. Если обе версии платят одним и тем же получателям, они будут оплачены дважды (и майнер получит комиссию за проведение двух отдельных транзакций).

Простой способ избежать этой проблемы – убедиться, что заменяющая транзакция всегда включает все те же входные данные, что и предыдущая версия транзакции.

Преимущество метода увеличения комиссии RBF перед другими типами заключается в том, что он позволяет очень эффективно использовать пространство блока. Часто заменяющая транзакция имеет тот же размер, что и заменяемая. Даже если она больше, ее размер часто равен размеру транзакции, которую пользователь создал бы, если бы изначально заплатил повышенную ставку комиссии.

Основным недостатком повышения комиссии RBF является возможность ее использования только создателем транзакции – человеком или людьми, которые должны были предоставить подписи или другие данные для аутентификации транзакции. Исключением являются транзакции, которые были разработаны с учетом возможности добавления дополнительных данных с помощью флагов sighash (см. раздел «Типы хеширования подписей (SIGHASH)»), но это сопряжено с определенными трудностями. В общем, если вы являетесь получателем неподтвержденной транзакции и хотите ее быстрее подтвердить (или вообще подтвердить), вам не удастся использовать RBF для увеличения комиссии; вам нужен другой метод.

Существуют дополнительные проблемы с RBF, которые будут рассмотрены в разделе «Закрепление транзакций».

Повышение комиссии Child Pays for Parent (CPFP)

Каждый получатель выхода неподтвержденной транзакции может мотивировать майнеров подтвердить эту транзакцию с использованием этого выхода. Требующая подтверждения транзакция называется *родительской транзакцией*

(*parent transaction*). Транзакция, которая тратит выход родительской транзакции, называется *дочерней транзакцией* (*child transaction*).

Как уже говорилось в разделе «Поле Outpoint», каждый вход в подтвержденной транзакции должен ссылаться на неизрасходованный выход предыдущей транзакции в блокчейне (в том же блоке или в предыдущем). Это означает, что майнер, который хочет подтвердить дочернюю транзакцию, должен также убедиться, что ее родительская транзакция уже подтверждена. Если родительская транзакция еще не подтверждена, но дочерняя транзакция платит достаточно высокую комиссию, майнер может рассмотреть возможность с выгодой для себя подтвердить сразу обе в одном блоке.

Для оценки выгодности майнинга родительской и дочерней транзакций майнер анализирует их как *пакет транзакций* (*package of transactions*) с суммарным размером и суммарной комиссией, на основании чего комиссию можно разделить на размер и вычислить *ставку комиссии пакета* (*package fee rate*). Затем майнер может отсортировать все известные ему отдельные транзакции и пакеты транзакций по размеру комиссии и включить самые выгодные из них в блок, который он пытается добыть, вплоть до максимального разрешенного размера (веса) для включения в блок. Чтобы найти еще больше выгодных для добычи пакетов, майнер может оценить эти пакеты в нескольких поколениях (например, неподтвержденную родительскую транзакцию объединить с дочерней и внучатой). Это называется *майнингом с оплатой по предкам* (*ancestor fee rate mining*).

В Bitcoin Core уже много лет реализован майнинг с оплатой по предкам. Предполагается, что на момент написания книги его используют почти все майнеры. Это означает, что кошельки могут использовать эту функцию для повышения комиссии за входящую транзакцию, используя дочернюю транзакцию для оплаты родительской (Child Transaction to Pay for its parent, CPFP).

Метод CPFP имеет ряд преимуществ перед RBF. CPFP может использовать любой, кто получает выход транзакции, то есть как получатель платежей, так и плательщик (если он включил выход изменений). Кроме того, CPFP не требует замены исходной транзакции, что делает этот метод менее опасным для некоторых продавцов, чем RBF.

Основным недостатком CPFP по сравнению с RBF считается увеличение занимаемого объема в блоке. В RBF транзакция с увеличением комиссии зачастую имеет тот же размер, что и заменяемая ею транзакция. В CPFP транзакция с увеличением комиссии добавляет совершенно новую транзакцию. Использование большего пространства в блоках требует дополнительной оплаты сверх расходов на увеличение комиссии.

Существует несколько проблем с CPFP, некоторые из которых будут рассмотрены в разделе «Закрепление транзакций». Проблема, о которой стоит упомянуть особо, – это вопрос минимальной ставки комиссии за передачу, который решается с помощью пакетной пересылки.

Пакетная пересылка

В ранних версиях Bitcoin Core не устанавливались ограничения на количество неподтвержденных транзакций в их временных хранилищах (мемпулах –

mempools) для последующей передачи и майнинга (см. раздел «Мемпулы и орфанные пулы»). Безусловно, компьютеры имеют физические ограничения, будь то память (RAM) или дисковое пространство. Поэтому полный узел не может хранить неограниченное количество неподтвержденных транзакций. В более поздних версиях Bitcoin Core размер мемпула был ограничен объемом транзакций примерно на один день, и в нем хранились только транзакции или пакеты с наибольшей комиссией.

В большинстве случаев это работает очень хорошо, но при этом возникает проблема зависимости. Чтобы рассчитать ставку комиссии для пакета транзакций, нужны как родительские, так и дочерние транзакции. Но притом, если родительская транзакция не платит достаточно высокую ставку комиссии, она не будет храниться в мемпуле узла. Если узел получает дочернюю транзакцию, не имея доступа к родительской, он не сможет с ней работать.

Решением этой проблемы является возможность передачи транзакций в виде пакета, так называемая *пакетная пересылка* (*package relay*), что позволяет принимающему узлу оценить размер комиссии всего пакета до начала работы с каждой отдельной транзакцией. На момент написания этой книги разработчики Bitcoin Core добились значительного прогресса в реализации пакетной пересылки, и к моменту публикации книги может быть доступна ее ограниченная ранняя версия.

Пакетная пересылка особенно важна для протоколов работы с чувствительными ко времени транзакциями с предварительной подписью, таких как Lightning Network (LN). В несогласованных случаях некоторые предварительно назначенные транзакции не могут быть переданы с помощью RBF, поэтому их приходится использовать с CPFP. Если предварительно подписанный транзакт платит комиссию ниже необходимой для попадания в мемпул узла суммы, у дочернего узла нет возможности поднять его комиссию. Если это помешает вовремя подтвердить транзакцию, добросовестный пользователь может потерять свои деньги. Решением этой важной проблемы является пакетная пересылка.

Закрепление транзакций

Несмотря на эффективность повышения комиссии с RBF и CPFP в описанных выше базовых случаях, для обоих методов существуют правила, призванные предотвратить атаки типа «отказ в обслуживании» на майнеров и передающие полные узлы. Неприятным побочным эффектом этих правил является вероятность блокировки использования функции повышения комиссии. Затруднение или отсутствие возможности использовать платную комиссию при проведении транзакций получило название «*закрепление транзакции*» (*transaction pinning*, часто также говорят «*пиннинг транзакции*»).

Одна из основных проблем, связанных с отказом в обслуживании, связана с эффектом взаимодействия транзакций. При расходовании выхода транзакции ее идентификатор (txid) упоминается дочерней транзакцией. Но при замене транзакции она получает другой txid. Если эта заменяющая транзакция получает подтверждение, ни один из ее потомков не сможет быть включен в тот же блокчейн. Можно пересоздать и заново подписать транзакции-потомки, но

это не дает никаких гарантий. Данный аспект имеет отношение к RBF и CPFP, но с разными последствиями:

- при использовании RBF, когда Bitcoin Core принимает заменяющую транзакцию, он все упрощает, поскольку позволяет «забыть» об исходной транзакции и всех последующих, которые зависели от исходной. Чтобы майнерам было выгоднее принимать замену, Bitcoin Core признает замену транзакции только в том случае, если она платит большую комиссию, нежели все «забытые» транзакции.

  Недостатком такого подхода является то, что Алиса может создать небольшую транзакцию для оплаты Бобу. Затем Боб может использовать свой выход для создания большой дочерней транзакции. Если Алиса впоследствии захочет заменить свою исходную транзакцию, ей придется заплатить за нее больше, чем она и Боб заплатили изначально. Например, если исходная транзакция Алисы была размером около 100 вбайт, а транзакция Боба – около 100 000 вбайт и они оба применяли одинаковые ставки комиссии, то теперь Алисе придется заплатить более чем в 1 тысячу раз больше, чем она заплатила первоначально, чтобы увеличить комиссию RBF по ее транзакции;
- при использовании CPFP каждый раз, когда узел рассматривает возможность включения пакета в блок, он должен удалить транзакции этого пакета из любого другого пакета, который он собирается рассмотреть для того же блока. Например, если дочерняя транзакция оплачивает 25 предков и у каждого из этих предков есть 25 других детей, то включение этого пакета в блок требует обновления примерно 625 пакетов (25^2). Аналогично если транзакция с 25 потомками удаляется из мемпула узла (например, за включение в блок), а у каждого из этих потомков есть 25 других предков, то придется обновить еще 625 пакетов. Каждый раз, когда этот параметр удваивается (например, с 25 до 50), объем выполняемой узлом работы увеличивается в четыре раза;
- кроме того, транзакцию и всех ее потомков не имеет смысла хранить в мемпуле в течение длительного времени, если добывается альтернативная версия этой транзакции – ведь теперь ни одна из них не может быть подтверждена, если только не произойдет редкий случай реорганизации блокчейна. Bitcoin Core будет удалять из своего мемпула все транзакции, которые больше не могут быть подтверждены на действующем блокчейне. В худшем случае это может привести к расходу огромного количества пропускной способности вашего узла и, возможно, будет потрачено на препятствование корректному распространению транзакций. Кроме того, транзакцию и всех ее потомков не имеет смысла хранить в мемпуле в течение длительного времени, если добывается альтернативная версия этой транзакции, – ведь теперь ни одна из них не может быть подтверждена, если только не произойдет редкий случай реорганизации блокчейна. Bitcoin Core будет удалять из своего мемпула все транзакции, которые больше не могут быть подтверждены на действующем блокчейне. В худшем случае это может привести к расходу огромного количества пропускной способности вашего узла и, возможно, будет потрачено на препятствование корректному распространению транзакций.

> Для предотвращения этих и других проблем Bitcoin Core ограничивает родительскую транзакцию максимум 25 предками или их потомками в своем мемпуле, а также ограничивает общий размер всех этих транзакций до 100 000 вбайт. Недостатком этого подхода является отсутствие у пользователей возможности создавать повышения комиссии CPFP у транзакции со слишком большим количеством потомков (или если она сама и ее потомки слишком велики).

Запрет транзакций может произойти случайно, но он также представляет собой серьезную уязвимость для многосторонних зависимых от времени протоколов, таких как LN. Если ваш партнер сможет предотвратить подтверждение одной из ваших транзакций к установленному сроку, он сможет украсть ваши деньги.

Разработчики протоколов уже несколько лет работают над смягчением проблем пиннинга транзакций. Одно из частичных решений описано в разделе «Исключение для CPFP и якорные выходы». Было предложено несколько других решений, и по крайней мере одно из них активно находилось в разработке на момент написания этой книги – эфемерные якоря (ephemeral anchors, https://bitcoinops.org/en/topics/ephemeral-anchors/).

Исключение для CPFP и якорные выходы

В 2018 году разработчики LN столкнулись с проблемой. Их протокол использует транзакции, требующие подписи двух разных сторон. Ни одна из сторон не собирается доверять другой, поэтому они подписывают транзакции в тот момент действия протокола, когда доверие не требуется, что позволяет одной из сторон передать одну из этих транзакций позже, когда другая сторона может оказаться не в состоянии (или не захочет) выполнить свои обязательства. Проблема такого подхода заключается в необходимости передачи транзакций в неопределенное время, далеко в будущем, за пределами разумной возможности оценить соответствующую ставку комиссии за эти транзакции.

Чисто теоретически разработчики могли бы предусмотреть в своих транзакциях возможность увеличения комиссии с помощью RBF (с использованием специальных флагов sighash) или CPFP, но оба этих протокола уязвимы к закреплению (пиннингу) транзакций. Поскольку участвующие транзакции критичны к временным показателям, разрешение контрагенту использовать пиннинг для задержки подтверждения транзакции может запросто стать причиной возникновения многократно повторяющейся уязвимости, которую злоумышленники могут использовать для кражи денег у добросовестных участников.

Разработчик LN Мэтт Коралло (Matt Corallo) предложил решение: сделать для правил повышения комиссии CPFP специальное разрешение под названием «*Исключение для CPFP*» (*CPFP carve out*). Стандартные правила CPFP запрещают включение дополнительного потомка, если в результате родительская транзакция будет иметь 26 или более потомков или если в результате размер родительской транзакции и всех ее потомков превысит 100 000 вбайт. В соответствии с исключением из правил CPFP в пакет может быть добавлена одна

дополнительная транзакция размером до 1000 вбайт, даже если она превышает другие ограничения, но при условии, что она является прямым потомком неподтвержденной транзакции без неподтвержденных предков.

Например, Боб и Мэллори совместно подписывают транзакцию с двумя выходами, по одному на каждого из них. Мэллори передает эту транзакцию и использует свой выход для добавления либо 25 дочерних транзакций, либо любого меньшего количества дочерних транзакций суммарным размером до 100 000 вбайт. Без исключения (carve-out) Боб не смог бы присоединить к своему выходу еще одну дочернюю транзакцию для повышения комиссии CPFP. С исключением из правил он может тратить принадлежащий ему один из двух выходов транзакции, если размер его дочерней транзакции не превышает 1000 вбайт (этого места должно быть более чем достаточно).

Не допускается использовать исключение для CPFP более одного раза, и поэтому оно работает только для двухсторонних протоколов. Были предложения расширить эту функцию на протоколы с большим числом участников, но особого спроса на это пока не наблюдается, и разработчики сосредоточены на создании других, более актуальных решений для защиты от атак с использованием пиннинга транзакций.

На данный момент большинство популярных реализаций LN используют шаблон транзакции под названием «*якорные выходы*» (*anchor outputs*), который рассчитан на применение совместно с функцией исключения для CPFP.

Добавление комиссии в транзакции

В структуре данных транзакций нет специального поля для комиссий. Вместо этого комиссия определяется как разница между суммой входов и суммой выходов. Любая лишняя сумма, которая остается после вычитания всех выходов из всех входов, и есть комиссия, взимаемая майнерами:

Комиссия = *Сумма*(*входов*) – *Сумма*(*выходов*).

Этот элемент транзакций несколько путаный, но понять его очень важно, поскольку при создании собственных транзакций вам нужно исключить возможность случайного включения очень большой комиссии из-за недостаточного расходования средств входа. Это означает, что нужно учитывать все входы и в случае необходимости вносить изменения, иначе в итоге майнерам достанутся очень большие чаевые!

Например, если вы тратите UTXO на 20 биткойнов для совершения платежа в 1 биткойн, вам необходимо вывести 19 биткойнов сдачи обратно на свой кошелек. В противном случае «остаток» в 19 биткойнов будет засчитан как комиссия за транзакцию и взыскан майнером, добывающим вашу транзакцию в блоке. Несмотря на приоритетную обработку транзакции и большую радость для майнера, это, скорее всего, не то, что вы планировали.

Если вы забудете добавить в созданную вручную транзакцию выход сдачи, вы заплатите ее в качестве комиссии за транзакцию. Возможно, это совсем не то, что вы подразумеваете под «оставьте сдачу себе!».

Блокировка по времени для защиты от перехвата комиссии

Сценарий перехвата комиссии (или «комиссионный снайпинг», fee sniping) представляет собой теоретический способ проведения атаки, при котором майнеры, которые пытаются переписать прошлые блоки, «перехватывают» (snipe) транзакции с более высокой комиссией из будущих блоков, чтобы максимизировать свою прибыльность.

К примеру, предположим, что самый высокий из существующих – это блок #100 000. И вместо майнинга блока #100 001 для продления цепочки некоторые майнеры пытаются повторно добыть #100 000. Эти майнеры могут включить любую корректную транзакцию (которая еще не была добыта) в свой блок-кандидат #100 000. Они не обязаны повторять блок с теми же транзакциями. По сути, у них есть стимул выбирать наиболее прибыльные (с наибольшей комиссией за килобайт) транзакции для включения в свой блок. Они могут включить любые транзакции, которые были в «старом» блоке #100 000, а также любые транзакции из актуального мемпула. То есть при повторном создании блока #100 000 у них есть возможность перетащить транзакции из «настоящего» в переписанное «прошлое».

Сегодня эта атака не очень выгодна, поскольку размер субсидии на блок значительно выше, чем общая комиссия за блок. Но в какой-то момент в будущем плата за транзакцию составит большую часть дохода (или даже весь доход). Тогда этот сценарий станет неизбежен.

В некоторых кошельках для предотвращения перехвата комиссии создаются транзакции со временем блокировки, которое позволяет ограничить включение этих транзакций в следующий или любой последующий блок. В нашем сценарии кошелек установит время блокировки 100 001 для любой созданной им транзакции. При обычных обстоятельствах это время блокировки не оказывает никакого влияния – транзакции в любом случае могут быть включены только в блок #100 001; ведь это следующий блок.

Однако при атаке с реорганизацией майнеры не смогут извлечь высокодоходные транзакции из мемпула, потому что все эти транзакции будут заблокированы по времени до блока #100 001. Они смогут повторно майнить блок #100 000 только теми транзакциями, которые были действительны на тот момент, и, по сути, не получат новой комиссии.

Это не позволяет полностью предотвратить «перехват» комиссий, но делает его в некоторых случаях менее прибыльным и может помочь сохранить стабильность сети Биткойн по мере снижения субсидий на блокчейн. Авторы рекомендуют всем кошелькам внедрять функцию защиты от перехвата комиссии, если она не мешает кошельку использовать блокировку по времени для других целей.

По мере развития Биткойна и сокращения субсидий комиссия становится все более важной и значимой для пользователей Биткойна как в повседневном использовании для быстрого подтверждения транзакций, так и в качестве мотивации майнеров к дальнейшему повышению безопасности транзакций Биткойна с помощью новых методов доказательства работы.

Глава 10

Сеть Биткойн

Биткойн представляет собой одноранговую сетевую архитектуру на основе интернета. Термин «одноранговая», или «равноправная», «пиринговая» (peer-to-peer, P2P), означает, что все полные узлы, участвующие в сети, равноправны относительно друг друга, и все они могут выполнять одни и те же функции, а каких-либо «особых» узлов не существует. Узлы сети соединяются в сеть с ячеистой (сотовой) «плоской» (flat) топологией. В сети нет ни сервера, ни централизованных служб, ни иерархии. Узлы P2P-сети предоставляют и потребляют услуги в одно и то же время. По своей природе P2P-сети устойчивы, децентрализованы и открыты. Ярким примером архитектуры P2P-сети был ранний интернет, где узлы IP-сети были равноправны. Архитектура современного интернета более иерархична, но интернет-протокол (Internet Protocol, IP) по-прежнему сохраняет свою плоскую топологию. Помимо Биткойна и интернета, самым крупным и успешным примером применения технологий P2P является файловый обмен: Napster был первопроходцем, а BitTorrent стал самым современным вариантом развития этой архитектуры.

Архитектура P2P-сети Биткойн – это нечто большее, чем просто выбор топологии. Биткойн – это по своей сути P2P-система цифровых денег, и архитектура сети является отражением и основой этой ключевой характеристики. Децентрализация управления – это основной принцип разработки, который может быть достигнут и поддерживаться только посредством плоской и децентрализованной P2P-сети консенсуса.

Под термином «сеть Биткойн» (Bitcoin network) подразумевают совокупность узлов, работающих по протоколу Bitcoin P2P. Помимо протокола Bitcoin P2P, существуют и другие, которые используются для майнинга и легких кошельков. Эти дополнительные протоколы предоставляются серверами маршрутизации шлюзов, которые получают доступ к сети Биткойн по протоколу Bitcoin P2P, а затем распространяют эту сеть на узлы, работающие по другим протоколам. Например, серверы Stratum подключают узлы майнинга Stratum через протокол Stratum к основной сети Биткойн и обеспечивают мостик между протоколом Stratum и P2P-протоколом Биткойна. В этой главе будут описаны некоторые из наиболее часто используемых протоколов в дополнение к базовому протоколу Bitcoin P2P.

Типы и роли узлов

Несмотря на то что полные узлы (full nodes, они же пиры, peers) в P2P-сети Bitcoin равноправны друг с другом, они могут играть разные роли в зависимости от поддерживаемых ими функций. Полный узел Биткойна проверяет блоки и может выполнять другие действия, такие как маршрутизация, майнинг и обслуживание кошельков.

Некоторые узлы, называемые *архивными полными узлами* (*archival full nodes*), также хранят полную и актуальную копию блокчейна. Эти узлы могут предоставлять данные тем клиентам, которые хранят только часть блокчейна и частично верифицируют транзакции с помощью метода под названием *упрощенная верификация платежей* (*simplified payment verification*, SPV). Их называют облегченными (легкими) клиентами (lightweight clients).

Майнеры конкурируют за создание новых блоков с помощью специализированного оборудования для выполнения алгоритма «доказательство работы» (proof-of-work). Некоторые майнеры работают как полноценные узлы, проверяя каждый блок в блокчейне, в то время как другие являются клиентами в пуле майнинга и зависят от сервера пула, обеспечивающего их работой.

Пользовательские кошельки могут подключаться к собственному полному узлу, как это иногда происходит с клиентами Биткойна для настольных компьютеров, однако многие пользовательские кошельки, особенно работающие на устройствах с ограниченными ресурсами, таких как смартфоны, являются облегченными узлами.

Помимо основных типов узлов протокола P2P Bitcoin, существуют серверы и узлы, работающие по другим протоколам, таким как специализированные протоколы майнинговых пулов и легкие протоколы клиентского доступа.

Сеть

На момент написания этой книги основная сеть Биткойн, в которой используется протокол Bitcoin P2P, насчитывала порядка 10 000 прослушивающих узлов на различных версиях Bitcoin Core и несколько сотен узлов на других реализациях протокола Bitcoin P2P, таких как BitcoinJ, btcd и bcoin. Небольшой процент узлов в сети Bitcoin P2P также занимается майнингом. Различные частные лица и компании взаимодействуют с сетью Биткойн, запуская архивные полные узлы с полными копиями блокчейна и сетевым узлом, но без функций майнинга и кошелька. Эти узлы выступают в качестве маршрутизаторов на границе сети, позволяя строить различные сервисы (биржи, кошельки, блокчейн, обработку торговых платежей) на их основе.

Компактная передача блоков

Как только майнер находит новый блок, он сообщает об этом в сеть Биткойн (в которую входят другие майнеры). Обнаруживший блок майнер может не-

медленно приступить к работе над ним; все остальные майнеры, которые еще не узнали об этом блоке, продолжают работать над предыдущим блоком до тех пор, пока не узнают о новом.

Если кто-то из этих майнеров создаст блок до того, как узнает о новом блоке, его блок будет конкурировать с новым блоком первого майнера. Только один из этих блоков будет включен в блокчейн, используемый всеми полными узлами, и майнеры получают деньги лишь за те блоки, которые получили общее признание.

Побеждает тот блок, на вершине которого первым будет построен второй блок (если нет другого почти равного). Это называется *гонкой поиска блоков* (*block-finding race*) и показано на рис. 10.1. Гонки поиска блоков дают преимущество крупнейшим майнерам, поэтому они действуют вразрез с важнейшим для Биткойна принципом децентрализации. Чтобы предотвратить гонки поиска блоков и позволить майнерам любых масштабов принимать равное участие в лотерее, которой является майнинг Биткойна, крайне важно минимизировать время между моментом анонса нового блока одним майнером и моментом получения этого блока другими майнерами.

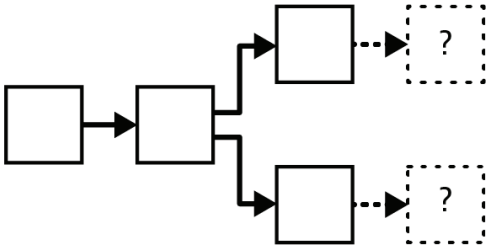


Рис. 10.1 ❖ Форк блокчейна, требующий проведения гонки майнинга

В 2015 году в новую версию Bitcoin Core была добавлена функция под названием *компактная передача блоков* (*compact block relay*), которая описана в BIP152. Она позволяет передавать новые блоки быстрее при меньшей нагрузке по пропускной способности.

Полные узлы, передающие неподтвержденные транзакции, также хранят многие из них в своих мемпулах (см. раздел «Мемпулы и орфанные пулы»). Когда некоторые из этих транзакций подтверждаются в новом блоке, этому узлу нет необходимости получать вторую копию данных транзакций.

Вместо получения избыточных неподтвержденных транзакций компактные блоки позволяют пирам отправлять короткий 6-байтовый идентификатор для каждой транзакции. Когда ваш узел получает компактный блок с одним или несколькими идентификаторами, он проверяет свой мемпул на наличие этих транзакций и задействует их в случае обнаружения. Для любой не найденной в мемпуле вашего локального узла транзакции ваш узел может отправить запрос пиру на ее получение.

И наоборот, если отдаленный пир считает, что в мемпуле вашего узла нет некоторых транзакций, которые появляются в блоке, он может включить копию этих транзакций в компактный блок. Например, Bitcoin Core всегда отправляет

в блок транзакцию coinbase. Если удаленный пир правильно угадал транзакции, которые есть в мемпуле вашего узла и которых у него нет, он отправит блок почти так же эффективно, как это теоретически возможно (для типичного блока эффективность составит от 97 до 99 %).

Компактная передача блоков не уменьшает их размер. Она просто предотвращает избыточную передачу уже имеющейся у узла информации. Если у узла ранее не было информации о блоке, например при первом запуске, он должен получить полные копии каждого блока.

В настоящее время Bitcoin Core поддерживает два режима отправки компактных блоков, как показано на рис. 10.2 (заштрихованная полоса показывает время, требуемое узлу для проверки блока).

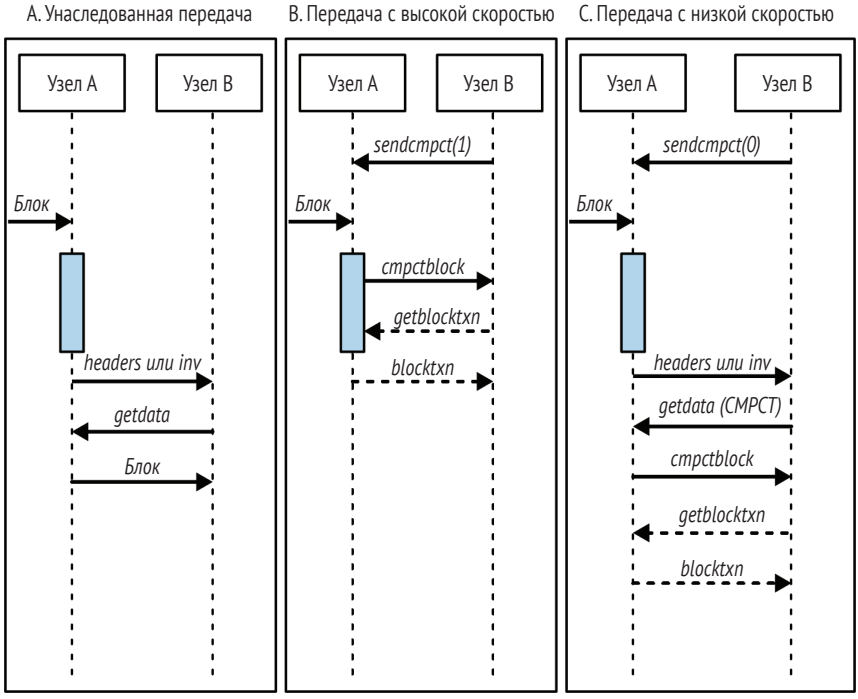


Рис. 10.2 ❖ Сравнение режимов BIP152

Особенности этих двух режимов отправки компактных блоков таковы.

Режим низкой пропускной способности

Когда ваш узел запрашивает у пира режим низкой пропускной способности (ow-bandwidth mode, используется по умолчанию), этот пир сообщает вашему узлу 32-байтовый идентификатор (хеш заголовка) нового блока, но не отправляет ему никаких подробностей. Если ваш узел сначала получит этот блок из другого источника, это позволит не тратить пропускную способность канала на получение избыточной копии этого блока. Если вашему узлу действительно нужен этот блок, он запросит компактный блок.

Режим высокой пропускной способности

Когда ваш узел просит у пира использовать режим высокой пропускной способности (high-bandwidth mode), этот пир отправит вашему узлу компактный блок для нового блока еще до того, как он проведет полную процедуру валидации этого блока. Единственная процедура валидации, которую выполнит пир, – это проверка наличия в заголовке блока правильного количества доказательств работы (proof of work). Поскольку генерация доказательства работы стоит недешево (примерно $150 000 на момент написания книги), маловероятно, что майнер будет подделывать его только ради пустой траты пропускной способности узлов передачи данных.

Пропуск процедуры валидации перед передачей позволяет новым блокам распространяться по сети с минимальными задержками на каждом узле. Недостатком режима высокой пропускной способности является то, что ваш узел, скорее всего, будет получать избыточную информацию от каждого выбранного им аналога с высокой пропускной способностью. На данный момент Bitcoin Core запрашивает только три пира для использования режима высокой пропускной способности (при этом он старается выбирать пиры с опытом быстрого анонсирования блоков).

Названия этих двух методов (взятые из BIP152) могут немного сбить с толку. Режим низкой пропускной способности снижает нагрузку на канал за счет того, что в большинстве случаев блоки не отправляются. Режим высокой пропускной способности использует большую пропускную способность, чем режим низкой пропускной способности, но в большинстве случаев гораздо меньшую, чем использовалась для передачи блоков до внедрения технологии компактных блоков.

Частные сети для передачи блоков

Помимо сокращения времени распространения блоков по сети, компактные блоки также позволяют значительно снизить задержки. В отличие от компактных блоков, другие варианты связаны с компромиссами, которые не подходят или не могут быть использованы в общедоступных сетях передачи данных P2P. По этой причине были проведены эксперименты с частными сетями передачи данных для блоков (private relay networks for blocks).

Одним из простейших вариантов является предварительный выбор маршрута между конечными точками. Например, сеть передачи данных с серверами, расположенными в центрах обработки данных вблизи крупных трансокеанских оптоволоконных линий, может пересылать новые блоки быстрее, чем если бы пришлось дожидаться поступления блока на узел домашнего пользователя, находящегося за много километров от оптоволоконной линии.

Другим, более сложным подходом является технология упреждающей коррекции ошибок (Forward Error Correction, FEC). Она дает возможность разделить компактное блочное сообщение на несколько фрагментов, к каждому из которых добавляются дополнительные (избыточные) данные. Если какой-то

из фрагментов не был получен, он может быть восстановлен из полученных. В зависимости от настроек можно восстановить до нескольких утерянных фрагментов.

Технология FEC предотвращает возникновение проблемы с отсутствием доставки компактных блоков (или некоторых их фрагментов) из-за проблем с основным сетевым соединением. Однако повторный запрос недостающих данных увеличивает время их получения втрое. Например:

1. Алиса отправляет Бобу определенные данные.
2. Боб не получает данные (или они повреждены). Боб заново запрашивает данные у Алисы.
3. Алиса снова отправляет данные.

Третий вариант – исходить из того, что все принимающие информацию узлы хранят в своих мемпулах почти одинаковые транзакции, поэтому все они могут принять один и тот же компактный блок. Это не только экономит время на вычисление компактного блока на каждом промежуточном узле, но и означает, что каждый промежуточный узел может просто передать пакеты FEC на следующий промежуточный узел даже до их проверки.

Основной проблемой каждого из вышеперечисленных способов является их эффективность в централизованной сети, но не в децентрализованной, где отдельные узлы не имеют возможности доверять другим узлам. Серверы в центрах обработки данных обходятся недешево и часто могут быть открыты для доступа операторам этого центра, что делает их менее надежными, чем защищенный домашний компьютер. Передача данных до проверки приводит к нерациональному расходованию трафика, поэтому такая схема может использоваться только в частных сетях, где существует определенный уровень доверия и взаимной ответственности сторон.

Первоначально сеть Bitcoin Relay Network https://www.bitcoinrelaynetwork.org/ была создана разработчиком Мэттом Коралло (Matt Corallo) в 2015 году для быстрой синхронизации блоков между майнерами с крайне низкой латентностью. Сеть состояла из нескольких виртуальных частных серверов (Virtual Private Server, VPS), расположенных на объектах инфраструктуры по всему миру. Она обеспечивала связь между большинством майнеров и майнинговых пулов.

На смену первоначальной сети Bitcoin Relay Network в 2016 году пришла сеть *Fast Internet Bitcoin Relay Engine*, или *FIBRE* https://bitcoinfibre.org/, также созданная Мэттом Коралло. Система FIBRE – это программа для работы с сетью передачи данных на основе протокола UDP, которая ретранслирует блоки по сети узлов. Система FIBRE использует технологию FEC и оптимизацию *компактных блоков* для дальнейшего сокращения объема передаваемых данных и задержек в сети.

Обнаружение сети

Каждый новый узел при загрузке должен отыскать в сети другие узлы Биткойна для начала совместной работы. Для запуска этого процесса новый узел дол-

жен обнаружить хотя бы один уже действующий узел в сети и подключиться к нему. Географическое расположение других узлов не имеет значения; топология сети Биткойн не имеет географических параметров. Поэтому любые уже действующие узлы Биткойна могут быть выбраны случайным образом.

Чтобы подключиться к одному из известных пиров, узлы устанавливают TCP-соединение, как правило, с портом 8333 (этот порт широко известен как используемый Биткойном), или с альтернативным ему портом, если таковой предоставлен. При создании соединения узел запускает «рукопожатие» («handshake», см. рис. 10.3) с передачей сообщения о версии, которое содержит основную идентифицирующую информацию, в том числе:

`Version`
: Версия протокола Bitcoin P2P, на которой «разговаривает» клиент (например, 70002).

`nLocalServices`
: Список поддерживаемых узлом локальных сервисов.

`nTime`
: Текущее время.

`addrYou`
: IP-адрес удаленного узла в том виде, в котором он отображается с этого узла.

`addrMe`
: IP-адрес локального узла, обнаруженный локальным узлом.

`Subvert`
: Подверсия, показывающая тип программы, запущенной на данном узле (например, `/Satoshi:0.9.2.1/`).

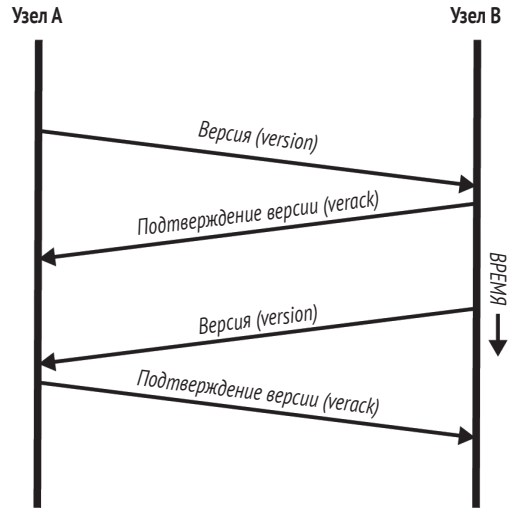


Рис. 10.3 ❖ Начальное «рукопожатие» между пирами

`BestHeight`
Высота блока в блокчейне этого узла.

`fRelay`
Поле, добавленное BIP37 для отказа в получении неподтвержденных транзакций.

Сообщение о версии всегда является самым первым сообщением, отправляемым любым пиром другому пиру. Локальный пир, получивший это сообщение, изучает информацию о версии удаленного пира и решает, совместим ли он с ним. Если удаленный пир совместим, локальный пир подтвердит сообщение версии и установит соединение, отправив сигнал `verack`.

Как новый узел находит пиры? Первый метод заключается в отправке DNS-запроса с использованием нескольких так называемых *DNS-сидов* (*DNS seed*), то есть DNS-серверов, которые предоставляют список IP-адресов узлов Биткойна. Некоторые из этих DNS-серверов предоставляют статический список IP-адресов стабильных узлов, прослушивающих Биткойн. Некоторые из DNS-сидов являются пользовательскими реализациями системы BIND (Berkeley Internet Name Daemon), которые возвращают случайное подмножество из списка адресов узлов Биткойна, собранного с помощью поисковой машины или давно работающего узла Биткойна. Клиент Bitcoin Core содержит имена нескольких различных DNS-сидов. Разнообразие владельцев и реализаций различных DNS-сидов обеспечивает высокий уровень надежности процесса начального запуска. В клиенте Bitcoin Core возможность использования DNS-сидов контролируется опциональным ключом `-dnsseed` (по умолчанию установлен на 1 для использования DNS-сидов).

Как вариант загружающемуся узлу, который еще ничего не знает о сети, необходимо предоставить IP-адрес хотя бы одного узла Биткойна, после чего он сможет устанавливать соединения путем установления дальнейших контактов. С помощью аргумента командной строки -seednode можно подключиться к одному узлу только для знакомства, используя его в качестве начального. После того как первоначальный узел-сид будет задействован для установления контактов, клиент отключится от него и будет использовать недавно обнаруженные пиры.

После установления одного или нескольких соединений новый узел отправляет своим пирам сообщение `addr` с его собственным IP-адресом. Пиры, в свою очередь, перешлют сообщение `addr` своим пирам, что обеспечит новому узлу широкую известность и улучшит его взаимодействие. Кроме того, вновь подключившийся узел может отправить пирам сообщение `getaddr`, запросив у них список IP-адресов других узлов. Таким образом, узел может находить пиры для подключения и сообщать о своем существовании в сети, чтобы его находили другие узлы. На рис. 10.4 показан протокол обнаружения адресов.

Чтобы создать различные маршруты выхода в сеть Биткойн, узел должен подключиться к нескольким различным пирам. Эти маршруты не являются стабильными – узлы появляются и исчезают, поэтому узел должен продолжать поиск новых узлов по мере утраты старых соединений, а также помогать другим узлам при их загрузке. Для начала работы необходимо только одно соеди-

нение, потому что первый узел может предложить знакомство своим пирам, а те, в свою очередь, могут предложить дальнейшие контакты. Также нет необходимости подключаться более чем к нескольким узлам и тратить сетевые ресурсы впустую. После загрузки узел запоминает последние успешные соединения с пирами, поэтому при перезагрузке он может быстро восстановить соединения с прежней сетью этих узлов. Если ни один из ранее подключенных узлов не отвечает на запрос о подключении, узел может использовать узлы-сиды для повторной начальной загрузки.

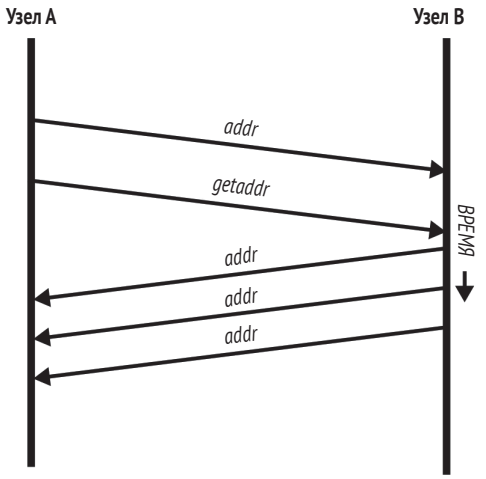


Рис. 10.4 ❖ Распространение и обнаружение адресов

На узле с работающим клиентом Bitcoin Core можно составить список соединений с пирами с помощью команды `getpeerinfo`:

```
$ bitcoin-cli getpeerinfo
[
  {
    "id": 0,
    "addr": "82.64.116.5:8333",
    "addrbind": "192.168.0.133:50564",
    "addrlocal": "72.253.6.11:50564",
    "network": "ipv4",
    "services": "0000000000000409",
    "servicesnames": [
      "NETWORK",
      "WITNESS",
      "NETWORK_LIMITED"
    ],
    "lastsend": 1683829947,
    "lastrecv": 1683829989,
    "last_transaction": 0,
    "last_block": 1683829989,
    "bytessent": 3558504,
    "bytesrecv": 6016081,
```

```
    "conntime": 1683647841,
    "timeoffset": 0,
    "pingtime": 0.204744,
    "minping": 0.20337,
    "version": 70016,
    "subver": "/Satoshi:24.0.1/",
    "inbound": false,
    "bip152_hb_to": true,
    "bip152_hb_from": false,
    "startingheight": 788954,
    "presynced_headers": -1,
    "synced_headers": 789281,
    "synced_blocks": 789281,
    "inflight": [
    ],
    "relaytxes": false,
    "minfeefilter": 0.00000000,
    "addr_relay_enabled": false,
    "addr_processed": 0,
    "addr_rate_limited": 0,
    "permissions": [
    ],
    "bytessent_per_msg": {
      ...
    },
    "bytesrecv_per_msg": {
      ...
    },
    "connection_type": "block-relay-only"
  },
]
```


Для отмены автоматического управления пирами и указания списка IP-адресов пользователи могут использовать опцию `-connect=<IPAddress>` и указать один или несколько IP-адресов. При использовании этой опции узел будет подключаться только к выбранным IP-адресам, а не находить и поддерживать соединения с пирами автоматически.

Если трафик на соединении отсутствует, узлы будут периодически отправлять сообщение для его обновления. Если узел не выходит на связь слишком долго, он считается отключенным, и начинается поиск нового пира. Таким образом, сеть динамически подстраивается под быстро изменяющиеся состояния узлов и возникающие проблемы, благодаря чему она может органично расти и сокращаться по мере необходимости без какого-либо центрального регулирования.

Полные узлы

Полными (полноценными) узлами (full nodes) называются те, которые проверяют каждую транзакцию в каждом блоке действующего блокчейна с максимальным доказательством работы (proof of work).

Полные узлы независимым образом обрабатывают каждый блок, начиная с самого первого (блок генезиса) и заканчивая последним известным блоком в сети. Полный узел может самостоятельно и полномочно проверить любую транзакцию. Полный узел полагается на сеть для получения обновлений о новых блоках транзакций, которые он затем проверяет и включает в свое локальное представление о тех скриптах, которые контролируют биткойны, что называют набором *выходов неизрасходованных транзакций* (*Unspent Transaction Outputs*, *UTXO*).

Запуск полного узла обеспечивает доступ к полному набору возможностей Биткойна: независимой верификации всех транзакций без необходимости полагаться на другие системы или доверять им.

Существует несколько альтернативных реализаций полных узлов с использованием различных языков программирования, программных архитектур и других конструктивных особенностей. Однако наиболее распространенной реализацией является Bitcoin Core. Более 95 % полных узлов в сети Биткойн работают на различных версиях Bitcoin Core. Она обозначается как «Satoshi» в строке subversion, которая отправляется в сообщении `version` и вызывается командой `getpeerinfo`, как было показано ранее; например, `/Satoshi:24.0.1/`.

Обмен «запасами»

Первым делом как только полный узел подключится к пирам, он попытается выстроить полную цепочку заголовков блоков. Если это совершенно новый узел без блокчейна, он знает только об одном блоке – блоке генезиса, который статически встроен в клиентскую программу. Теперь начиная с блока #0 (блок генезиса) новому узлу придется загрузить сотни тысяч блоков для синхронизации с сетью и воссоздания полного блокчейна.

Процесс синхронизации блокчейна начинается с сообщения о версии, поскольку в нем содержится параметр `BestHeight` – текущая высота блокчейна узла (количество блоков). Узел видит сообщения о версии от своих пиров, он узнает о количестве блоков у каждого из них и может сравнить его со своим собственным блокчейном. Пиринговые узлы обмениваются сообщением `getheaders`, содержащим хеш верхнего блока в их локальном блокчейне. Кто-то из пиров сможет идентифицировать полученный хеш как принадлежащий блоку, который не находится на вершине, а является более старым блоком, и таким образом сделать вывод, что его собственный локальный блокчейн длиннее, чем блокчейн другого узла.

Пир с более длинным блокчейном имеет больше блоков, чем другой узел, и он может определить заголовки, которые нужны другому узлу, чтобы «догнать» его. Он определит первые 2000 заголовков для обмена с помощью сообщения `headers`. Узел будет продолжать запрашивать дополнительные заголовки до тех пор, пока не получит по одному для каждого блока, о наличии которого заявляет удаленный пир.

Одновременно с этим узел начнет запрашивать блоки для каждого ранее полученного заголовка с помощью сообщения `getdata`. Узел будет запрашивать

разные блоки у каждого из выбранных им пиров, что позволит ему отказаться от соединений со значительно более медленными, чем в среднем, пирами, чтобы найти более новые (и, возможно, более быстрые) пиры.

Представим, что в распоряжении узла есть только блок генезиса. Затем он получит от своих пиров сообщение `headers` с заголовками следующих 2000 блоков в цепочке. Он начнет запрашивать блоки у всех своих подключенных пиров, сохраняя в очереди до 1024 блоков. Валидация блоков должна происходить по порядку. Если самый старый блок в очереди является тем, который должен быть подтвержден следующим, но еще не был получен, узел разрывает соединение с пиром, который должен был предоставить этот блок. Затем он находит новый пир, который может предоставить один блок до того, как все остальные пиры этого узла смогут предоставить ему 1023 блока.

По мере получения каждого блока он добавляется в блокчейн, как будет показано в главе 11. По мере наращивания локального блокчейна запрашиваются и принимаются дополнительные блоки. Этот процесс продолжается до тех пор, пока узел не догонит остальных участников сети.

Данный процесс сопоставления локального блокчейна с пирами и последующее получение недостающих блоков происходят каждый раз, когда узел находится вне сети в течение продолжительного периода времени.

Облегченные клиенты

Многие клиенты Биткойна предназначены для работы на устройствах ограниченного размера и мощности, таких как смартфоны, планшеты или встраиваемые системы. В таких устройствах применяют метод *упрощенной верификации платежей* (*Simplified Payment Verification*, SPV), чтобы они могли действовать без валидации полного блокчейна. Эти типы устройств называют облегченными (легкими) клиентами (lightweight clients).

Облегченные клиенты загружают только заголовки блоков и не загружают сами транзакции, входящие в каждый блок. Полученная цепочка заголовков без транзакций примерно в 10 000 раз меньше, чем полный блокчейн. Облегченные клиенты не могут составить полную картину всех UTXO, доступных для расходования, поскольку не располагают информацией о всех транзакциях в сети. Поэтому они проверяют транзакции с помощью немного другого метода с привлечением пиров, которые по запросу могут предоставить частичное описание соответствующих фрагментов блокчейна.

В качестве аналогии полный узел можно сравнить с туристом в незнакомом городе, у которого есть подробная карта всех улиц и всех адресов. В сравнении с этим облегченный клиент похож на туриста в незнакомом городе, который знает только один главный проспект и спрашивает у случайных прохожих дорогу от одного поворота к другому. Хотя оба туриста могут подтвердить существование улицы после ее посещения, турист без карты не имеет представления о происходящем на любой соседней улице, как и не знает о существовании других улиц. Находясь перед домом 23 по Черч-стрит, турист без карты не имеет ни малейшего представления о том, есть ли в городе еще десяток

таких же адресов «Черч-стрит, 23» и искомый ли это адрес. Лучшим выбором для туриста без карты будет наведение справок у достаточного количества людей и надежда на то, что кто-нибудь из них не окажется грабителем.

Облегченные клиенты проверяют транзакции по их *глубине* (*depth*) в блокчейне. Если полный узел выстраивает полностью проверенную цепочку из тысяч блоков и миллионов транзакций, идущую вниз по блокчейну (назад во времени) вплоть до блока генезиса, то облегченный клиент проверяет доказательства работы всех блоков (но не достоверность блоков и всех их транзакций) и связывает эту последовательность с интересующей транзакцией.

Например, при проверке транзакции в блоке 800 000 полный узел проверяет все 800 000 блоков вплоть до блока генезиса и создает полную базу данных UTXO, определяя истинность транзакции, подтверждая ее существование и наличие неизрасходованных выходов. Облегченный клиент может только подтвердить существование транзакции. Он устанавливает связь между транзакцией и содержащим ее блоком с помощью *пути Меркла* (*merkle path*, см. раздел «Деревья Меркла»). Затем облегченный клиент дожидается, пока не увидит шесть блоков с 800 001 по 800 006, собранных поверх блока с этой транзакцией, и верифицирует его, определяя глубину под блоками с 800 006 по 800 001. Тот факт, что другие узлы сети приняли блок 800 000, а майнеры сделали все необходимое для создания еще шести блоков поверх него, является косвенным доказательством реального существования транзакции.

Как правило, облегченный клиент не может убедиться в существовании транзакции в блоке, если на самом деле транзакция не существует. Облегченный клиент устанавливает существование транзакции в блоке, запрашивая доказательство пути Меркла и проверяя доказательство работы в цепочке блоков. Однако существование транзакции может быть «скрыто» от облегченного клиента. Он может проверить существование транзакции, но не может удостовериться в том, что транзакция, например, дважды расходующая один и тот же UTXO, не существует, поскольку у него нет записей обо всех транзакциях. Эта уязвимость может быть использована для атаки типа «отказ в обслуживании» (denial-of-service attack) или «двойная трата» (double-spending) против облегченных клиентов. Для защиты от этого такой клиент должен случайным образом подключаться к нескольким другим клиентам. Это увеличивает вероятность установления контакта хотя бы с одним добросовестным узлом. Необходимость случайного подключения означает, что облегченные клиенты также уязвимы к атакам разделения сети на фрагменты (network partitioning attacks) или атакам Сибиллы (Sybil attacks), когда они подключаются к поддельным узлам или поддельным сетям без доступа к настоящим узлам или реальной сети Биткойн.

Во многих практических ситуациях правильно подключенные легкие клиенты достаточно надежны, что обеспечивает баланс между потребностями в ресурсах, практичностью и безопасностью. Однако для безупречной безопасности нет ничего лучше, чем запуск полноценного узла.

Полноценный узел проверяет транзакцию по всей последовательности тысяч блоков под ней, чтобы гарантировать существование и нерастраченность UTXO, в то время как облегченный клиент доказывает только существование транзакции и устанавливает, что блок с этой транзакцией погребен под горсткой блоков над ним.

Для получения заголовков блоков, необходимых для подтверждения принадлежности транзакции к определенной цепочке, облегченные клиенты используют сообщение `getheaders`. Отвечающий пир отправит до 2000 заголовков блоков с помощью одного сообщения `headers`. См. рис. 10.5.

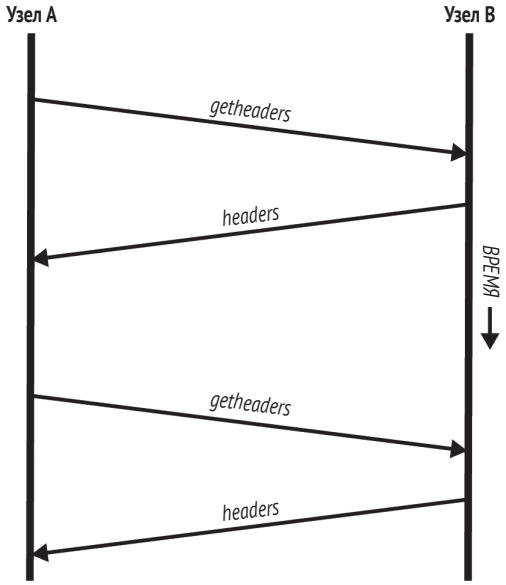


Рис. 10.5 ❖ Синхронизация заголовков блоков облегченным клиентом

Заголовки блоков дают облегченному клиенту возможность проверить соответствие отдельного блока блокчейну с наиболее весомым доказательством работы, но они не сообщают клиенту о содержащихся в блоках транзакциях, представляющих интерес для его кошелька. Клиент мог бы загружать и проверять каждый блок, но это потребовало бы значительного расхода ресурсов, необходимых для запуска полноценного узла. Поэтому разработчики искали другие пути решения проблемы.

Вскоре после внедрения облегченных клиентов разработчики Биткойна добавили функцию, получившую название *фильтры Блума* (*bloom filters*), с целью снизить нагрузку на пропускную способность канала, необходимую облегченным клиентам для получения информации о входящих и исходящих транзакциях. Фильтры Блума помогают облегченным клиентам получать подмножество транзакций, не раскрывая напрямую интересующие их адреса, за счет механизма фильтрации с использованием вероятностей, а не фиксированных шаблонов.

Фильтры Блума

Фильтр Блума – это вероятностный фильтр поиска, способ описать желаемый шаблон без его точного определения. Фильтры Блума дают эффективный способ задать шаблон поиска с сохранением конфиденциальности. Они исполь-

зуются облегченными клиентами для запроса у своих пиров транзакций по определенному шаблону, не сообщая при этом информации о конкретных разыскиваемых ими адресах, ключах или транзакциях.

В предыдущей рассмотренной нами аналогии турист без карты спрашивал дорогу к конкретному адресу, «Черч-стрит, 23». Если он спросит дорогу у незнакомца, он невольно выдаст свой пункт назначения. Фильтр Блума похож на вопрос «есть ли в этом районе улицы, название которых заканчивается на Р-Ч?». Такой вопрос раскрывает чуть меньше информации о нужном пункте назначения, чем запрос «Черч-стрит, 23». Используя этот прием, турист может указать желаемый адрес более подробно, например «заканчивающиеся на Е-Р-Ч», или менее подробно, например «заканчивающиеся на Ч». Варьируя точность поиска, турист раскрывает больше или меньше информации за счет получения более или менее конкретных результатов. Если турист задает менее точный шаблон, он получает гораздо больше вероятных адресов и лучшую конфиденциальность, но многие результаты оказываются нерелевантными. Если турист задает очень конкретный шаблон, он получает меньше результатов, но теряет конфиденциальность.

Фильтры Блума выполняют эту функцию, позволяя облегченному клиенту задать шаблон поиска транзакций с возможностью настройки на точность или конфиденциальность. Более точный фильтр Блума дает самые верные результаты, но за счет раскрытия интересующих клиента шаблонов, что приводит к раскрытию адресов кошелька пользователя. Менее узкий фильтр Блума будет выдавать больше данных о большем количестве транзакций, многие из которых не имеют отношения к клиенту, но позволит ему сохранить большую конфиденциальность.

Принцип работы фильтров Блума

Фильтры Блума состоят из массива двоичных цифр переменного размера N (битового поля) и хеш-функций переменного размера M. Хеш-функции созданы таким образом, чтобы на выходе всегда получалось значение от 1 до N, соответствующее массиву двоичных цифр. Хеш-функции генерируются детерминированно, поэтому любой клиент с фильтром Блума всегда будет использовать одни и те же хеш-функции и получать одни и те же результаты для определенного входа. Выбирая фильтры Блума разной длины (N) и различное количество хеш-функций (M), можно настраивать фильтр Блума, регулируя уровень точности и, соответственно, конфиденциальности.

На рис. 10.6 для демонстрации работы фильтров Блума используется очень маленький массив из 16 бит и набор из трех хеш-функций.

Изначально фильтр Блума создается с массивом битов в виде сплошных нулей. Чтобы добавить шаблон в фильтр Блума, он поочередно хешируется каждой хеш-функцией. При использовании первой хеш-функции на входе получается число от 1 до N. Соответствующий бит в массиве (проиндексированный от 1 до N) находится и устанавливается в 1, таким образом записывается выход хеш-функции. Затем следующая хеш-функция используется для установки другого бита и т. д. После применения всех M хеш-функций шаблон поиска будет «записан» в фильтр Блума в виде M бит, измененных с 0 на 1.

Рис. 10.6 ❖ Пример упрощенного фильтра Блума

На рис. 10.7 приведен пример добавления шаблона «A» в простой фильтр Блума из рис. 10.6.

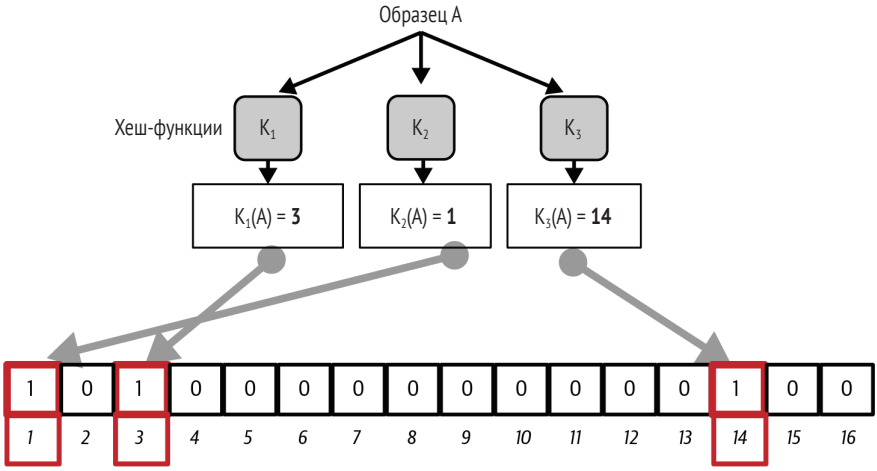


Рис. 10.7 ❖ Добавление шаблона «A» в простой фильтр Блума

Добавление второго шаблона – это просто повтор этого процесса. Шаблон хешируется каждой хеш-функцией по очереди, и результат записывается путем установки битов в 1. Обратите внимание, что по мере заполнения фильтра Блума большим количеством шаблонов результат хеш-функции может совпасть с битом, который уже установлен в 1, и в этом случае бит не изменяется. По сути, по мере записи все большего количества шаблонов в перекрывающиеся биты фильтр Блума начинает насыщаться все большим количеством битов, установленных в 1, и точность фильтра снижается. Именно поэтому фильтр и является вероятностной структурой данных – он становится менее точным по мере добавления большего количества шаблонов. Точность зависит от количества добавленных шаблонов в зависимости от размера битового массива (N) и количества хеш-функций (M). Большой битовый массив и большее количество хеш-функций могут записать больше деталей с высокой точностью. Мень-

ший битовый массив или меньшее количество хеш-функций запишут меньше деталей и дадут меньшую точность.

На рис. 10.8 показан пример добавления второго шаблона «B» к простому фильтру Блума.

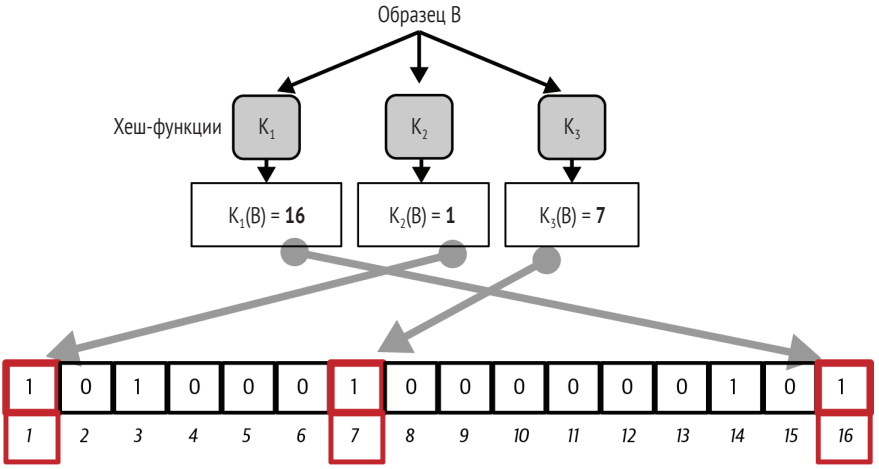


Рис. 10.8 ❖ Добавление второго шаблона «B» в простой фильтр Блума

Для проверки принадлежности шаблона к фильтру Блума шаблон хешируется каждой хеш-функцией, а полученный битовый шаблон проверяется на соответствие битовому массиву. Если все биты, индексированные хеш-функциями, установлены в `1`, то паттерн, *возможно*, записан в фильтр Блума. Поскольку биты могут быть установлены из-за наложения нескольких шаблонов, ответ не является однозначным, а носит, скорее, вероятностный характер. Проще говоря, положительное совпадение фильтра Блума – это «Возможно, да» (Maybe, yes).

На рис. 10.9 приведен пример проверки существования шаблона «X» в простом фильтре Блума. Соответствующие биты установлены в `1`, поэтому шаблон, вероятно, совпадает. В итоге получается вероятностное положительное совпадение, означающее «Возможно» (Maybe).

Напротив, если шаблон проверяется на соответствие фильтру Блума и любой из битов принимает значение `0`, это доказывает, что шаблон не был занесен в фильтр Блума. Отрицательный результат – это не вероятность, это определенность. Проще говоря, отрицательное совпадение с фильтром Блума – это «Определенно нет!».

На рис. 10.10 приведен пример проверки существования шаблона «Y» в простом фильтре Блума. Один из соответствующих битов установлен в `0`, поэтому паттерн определенно не совпадает. Результат – окончательное отрицательное совпадение, что означает «Определенно нет!».

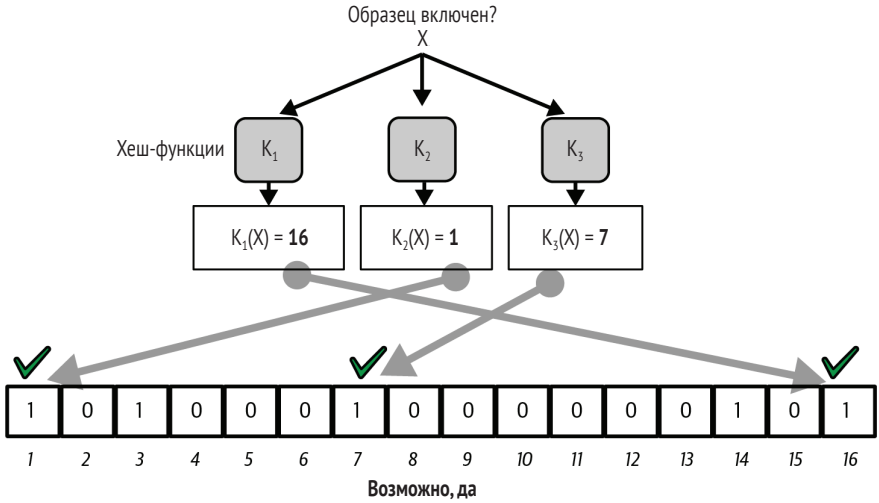


Рис. 10.9 ❖ Проверка существования шаблона «X» в фильтре Блума

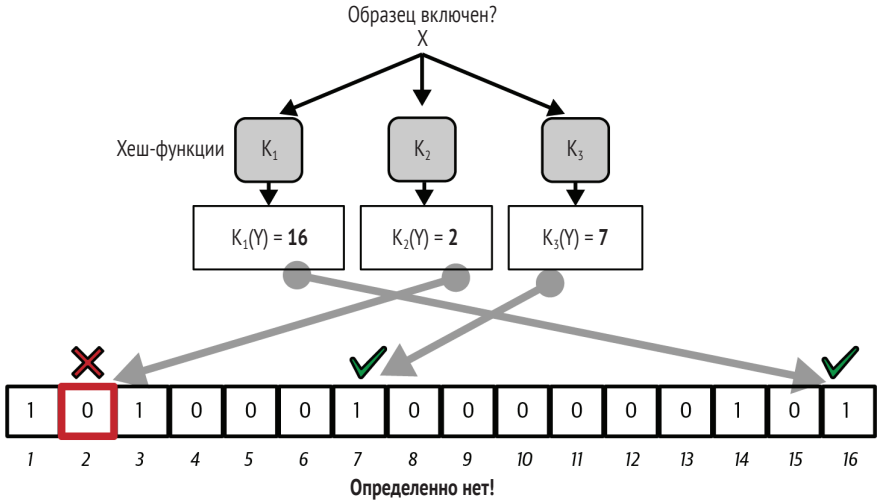


Рис. 10.10 ❖ Проверка существования шаблона «Y» в фильтре Блума

Использование фильтров Блума облегченными клиентами

Фильтры Блума используются для сортировки транзакций (и содержащих их блоков), которые облегченный клиент получает от своих пиров, выбирая только интересующие его транзакции, не раскрывая, какие именно адреса или ключи его интересуют.

Облегченный клиент инициализирует фильтр Блума как «пустой»; в этом состоянии он не будет соответствовать никаким шаблонам. Затем облегченный клиент составит список всех адресов, ключей и хешей, которые его интересуют. Для этого он извлекает хеш открытого ключа, хеш скрипта и идентификаторы транзакций из всех UTXO, контролируемых его кошельком. Затем облегченный клиент добавляет каждый из них в фильтр Блума, чтобы тот «совпал», если эти шаблоны присутствуют в транзакции, но при этом не раскрывая самих шаблонов.

Потом облегченный клиент отправляет пиру сообщение filterload с фильтром Блума для использования в этом соединении. У пира фильтры Блума проверяются на каждой входящей транзакции. Полный узел проверяет несколько частей транзакции с помощью фильтра Блума, выискивая совпадения, в частности:

- идентификатор транзакции;
- компоненты данных из скриптов каждого из выходов транзакций (каждый ключ и хеш в скрипте);
- каждый из входов транзакции;
- каждый из компонентов данных подписи входа (или скриптов свидетелей).

Проверяя все эти компоненты, фильтры Блума могут быть использованы для сопоставления хешей открытых ключей, скриптов, значений `OP_RETURN`, открытых ключей в подписях или любых других компонентов смарт-контракта либо сложного скрипта.

После создания фильтра пир будет проверять выходы каждой транзакции на совпадение с фильтром Блума. Клиенту отправляются только те транзакции, которые совпали с фильтром.

В ответ на сообщение `getdata` от клиента пиры отправляют сообщение `merkleblock`, содержащее только соответствующие фильтру заголовки блоков и путь Меркла (см. «Деревья Меркла») для каждой соответствующей транзакции. Пир также отправит сообщения `tx` с транзакциями, соответствующими фильтру.

По мере отправки полным узлом транзакций облегченному клиенту последний отбрасывает все ложные результаты и использует правильные транзакции для обновления своего набора UTXO и баланса кошелька. По мере обновления своего набора UTXO он также изменяет фильтр Блума для соответствия всем будущим транзакциям, ссылающимся на только что найденный UTXO. Затем полный узел использует новый фильтр Блума для сопоставления новых транзакций, и весь процесс повторяется.

Клиент с фильтром Блума может интерактивно добавлять шаблоны в фильтр с помощью отправки сообщения `filteradd`. Для очистки фильтра Блума клиент может отправить сообщение `filterclear`. Поскольку удалить шаблон из фильтра Блума невозможно, клиенту придется очистить и повторно отправить новый фильтр Блума после прекращения использования шаблона.

Сетевой протокол и принцип работы фильтра Блума для облегченных клиентов определены в BIP37.

К сожалению, после внедрения фильтров Блума стало ясно, что они не обеспечивают высокой степени конфиденциальности. Полный узел, получивший

фильтр Блума от пира, может применить этот фильтр ко всему блокчейну для поиска всех транзакций клиента (плюс ложные срабатывания). Затем он может искать закономерности и взаимосвязи между транзакциями. Случайно выбранные ложноположительные транзакции вряд ли будут иметь взаимосвязь «родительский–дочерний» от выхода к входу, но транзакции из кошелька пользователя с большой вероятностью могут иметь эту привязку. Если все связанные транзакции имеют определенные характеристики, например хотя бы один выход P2PKH, то можно предположить, что транзакции без этих характеристик не относятся к кошельку.

Также выяснилось, что специально созданные фильтры могут заставить обрабатывающие их узлы выполнять большой объем работы, что может стать причиной атак типа «отказ в обслуживании».

Именно из-за этих двух причин Bitcoin Core в конечном итоге ограничил поддержку фильтров Блума только клиентами с IP-адресами, которые прямо разрешены оператором узла. Таким образом, возникла необходимость в создании альтернативного метода для оказания помощи облегченным клиентам в поиске их транзакций.

Компактные фильтры блоков

В 2016 году анонимный разработчик опубликовал в списке рассылки Bitcoin-Dev идею обратного процесса фильтра Блума. При использовании фильтра Блума из протокола BIP37 каждый клиент хеширует свой адрес для создания фильтра Блума, а узлы хешируют части каждой транзакции для проверки соответствия этому фильтру. В новом решении узлы хешируют части каждой транзакции в блоке для создания фильтра Блума, а клиенты хешируют свои адреса, пытаясь найти соответствие этому фильтру. Если клиент находит совпадение, он загружает весь блок.

Несмотря на схожесть названий, *компактные блоки* (*compact blocks*) BIP152 и *фильтры компактных блоков* (*compact block filters*) BIP157/158 не связаны между собой.

Это позволяет узлам создавать единый фильтр для каждого блока, который они могут сохранять на диске и обслуживать снова и снова, устраняя уязвимость от атак типа «отказ в обслуживании» в BIP37. Клиенты не сообщают полным узлам никакой информации о своих прошлых или будущих адресах. Они только загружают блоки, которые могут содержать тысячи транзакций, не созданных клиентом. Они даже могут загружать каждый подходящий блок от разных пиров, что усложняет полным узлам возможность сопоставления транзакций одного клиента в нескольких блоках.

Эта идея с создаваемыми сервером фильтрами не обеспечивает идеальной конфиденциальности; она по-прежнему налагает определенные расходы на полные узлы (и требует увеличения пропускной способности облегченных клиентов для загрузки блоков), а фильтры могут использоваться только для подтвержденных транзакций (но не для неподтвержденных). Тем не менее эта методика намного более конфиденциальна и надежна, чем запрашиваемые клиентами фильтры Блума из BIP37.

После описания исходной идеи на основе фильтров Блума разработчики пришли к выводу, что есть лучшая структура данных для генерируемых сервером фильтров, которая получила название кодированные множества Голомба–Райса (Golomb-Rice Coded Sets, GCS).

Кодированные множества Голомба–Райса (GCS)

Представим, что Алиса хочет отправить Бобу список чисел. Простой способ сделать это – переслать ему сразу весь список этих чисел:

```
849
653
476
900
379
```

Но есть и более эффективный способ. Сначала Алиса располагает список в порядке возрастания:

```
379
476
653
849
900
```

Затем Алиса отправляет первое число. Для остальных чисел она отправляет разницу между этим числом и предыдущим. Например, для второго числа она отправляет 97 (476 – 379); для третьего – 177 (653 – 476) и т. д.:

```
379
97
177
196
51
```

Как видно, разность между двумя числами в упорядоченном списке дает числа, которые короче исходных. После получения Боб может восстановить исходный список, просто сложив каждое число с его предшественником. Это позволяет сэкономить место без потери информации, что и называется *кодированием без потерь* (*lossless encoding*).

Если случайным образом выбрать числа в фиксированном диапазоне значений, то чем больше чисел будет выбрано, тем меньше будет средняя разница. Это означает, что объем передаваемых данных растет не так быстро, как увеличивается длина списка (до определенного момента).

Более того, длина случайно выбранных чисел в списке различий естественным образом смещена в сторону меньшей длины. Возьмем два случайных числа от 1 до 6; это то же самое, что бросить две игральные кости. Существует 36 различных комбинаций двух игральных костей:

```
1 1  1 2  1 3  1 4  1 5  1 6
2 1  2 2  2 3  2 4  2 5  2 6
3 1  3 2  3 3  3 4  3 5  3 6
```

```
4 1  4 2  4 3  4 4  4 5  4 6
5 1  5 2  5 3  5 4  5 5  5 6
6 1  6 2  6 3  6 4  6 5  6 6
```

Найдем разность между большим и меньшим числами:

```
0  1  2  3  4  5
1  0  1  2  3  4
2  1  0  1  2  3
3  2  1  0  1  2
4  3  2  1  0  1
5  4  3  2  1  0
```

Если подсчитать частоту встречаемости каждой разницы, можно увидеть, что маленькие разности встречаются гораздо чаще, чем большие, см. табл. 10.1:

Таблица 10.1. Частота встречаемости разницы

Разница	Частота встречаемости
0	6
1	10
2	8
3	6
4	4
5	2

Если известно, что для хранения больших чисел может понадобиться большой объем (потому что большие разности случаются, хотя и редко), но чаще всего придется хранить небольшие числа. Каждое число можно закодировать с помощью системы, которая использует меньше места для небольших чисел и больше места для больших. В среднем такая система будет работать лучше, чем при использовании одинакового количества места для каждого числа.

Такую возможность предоставляет кодирование Голомба. Кодирование Райса – это подмножество кодирования Голомба, которое удобнее для использования в некоторых ситуациях, в том числе в случае применения фильтров блоков Биткойна.

Какие данные включать в фильтр блоков

Наша главная цель – обеспечить кошелькам возможность определять наличие в блоке транзакций, имеющих отношение к этому кошельку. Для эффективной работы кошелька ему необходимо знать два вида информации:

Когда он получил деньги

В частности, когда транзакция на выходе содержит контролируемый кошельком скрипт (например, с помощью авторизованного секретного ключа).

Когда он потратил деньги

В частности, когда вход транзакции ссылается на предыдущий выход транзакции, который кошелек контролировал.

Второстепенной задачей при разработке компактных фильтров блоков было создание условий, для того чтобы получающий фильтр кошелек мог удостовериться в получении корректного фильтра от своего пира. Например, загрузив блок, на основе которого был создан фильтр, кошелек мог сгенерировать свой собственный фильтр. Затем он мог сравнить свой фильтр с фильтром пира и подтвердить, что они идентичны, тем самым подтверждая факт создания пиром корректного фильтра.

Для достижения главной и второстепенной целей фильтр должен ссылаться на два вида информации:

- скрипт для каждого выхода в каждой транзакции в блоке;
- поле outpoint для каждого входа в каждой транзакции в блоке.

Первоначальный вариант компактных фильтров блоков включал в себя обе эти части информации, однако выяснилось, что есть более эффективный способ достижения главной цели, если пожертвовать второстепенной. В новом проекте фильтр блоков по-прежнему будет ссылаться на два типа информации, но уже в более тесной взаимосвязи:

- как и раньше, скрипт для каждого выхода в каждой транзакции в блоке;
- при изменении он также будет ссылаться на скрипт выхода, на который ссылается поле outpoint, для каждого входа в каждой транзакции в блоке. Иными словами, скрипт расходуемого выхода.

Такой подход имел несколько преимуществ. Во-первых, кошелькам не нужно было отслеживать точки выхода; вместо этого они могли просто сканировать скрипты выхода, на которые они ожидали получить деньги. Во-вторых, если последующая транзакция в блоке расходует средства, полученные от предыдущей транзакции в том же блоке, они обе будут ссылаться на один и тот же скрипт выхода. Более одной ссылки на один и тот же скрипт выхода является излишеством в компактном фильтре блоков, поэтому лишние копии могут быть удалены, что уменьшает размер фильтров.

Когда полные узлы проверяют блок, им нужен доступ к скриптам выхода как для текущих выходов транзакций в блоке, так и для выходов транзакций из предыдущих блоков, на которые ссылаются входы. Поэтому они могут строить компактные фильтры блоков по этой упрощенной модели. Но сам блок не включает скрипты выхода транзакций из предыдущих блоков, поэтому у клиента нет удобного способа проверить корректность построения фильтра блока. Однако есть альтернатива, которая может дать клиенту возможность обнаружить, что пир его обманывает за счет получения одного и того же фильтра от нескольких пиров.

Загрузка фильтров блоков от нескольких пиров

Пир может предоставить кошельку некорректный фильтр. Существует два способа создания неправильного фильтра. Пир может создать фильтр со ссылкой на транзакции, которые на самом деле не содержатся в соответствующем блоке (ложное срабатывание). Или же пир может создать фильтр, который не будет ссылаться на транзакции, действительно присутствующие в соответствующем блоке (ложноотрицательный фильтр).

Первым способом защиты от неточного фильтра является получение клиентом фильтра от нескольких пиров. Протокол BIP157 позволяет клиенту загрузить лишь короткое 32-байтовое обязательство фильтра для определения соответствия фильтра, предлагаемого каждым пиром, тем же самым фильтрам от всех других пиров клиента. Это уменьшает расход трафика, необходимого клиенту для запроса фильтров у множества различных пиров – при условии согласия всех этих пиров.

Если два или более разных пиров имеют разные фильтры для одного и того же блока, клиент может загрузить их все. Затем он также может загрузить соответствующий блок. Если в блоке содержится какая-либо связанная с кошельком транзакция, не входящая ни в один из фильтров, кошелек получает все основания быть уверенным, что пир создал ошибочный фильтр – кодовые наборы Голомба–Райса всегда будут содержать потенциальное совпадение.

С другой стороны, если в блоке нет транзакции, которая, по данным фильтра, могла бы соответствовать кошельку, это не является доказательством ошибочности фильтра. Чтобы минимизировать размер GCS, допускается определенное количество ложных срабатываний. Кошелек может и дальше загружать дополнительные фильтры от пира – либо случайным образом, либо при обнаружении совпадений, а затем отслеживать процент ложных срабатываний клиента. Если он значительно превышает показатель ложных срабатываний для таких фильтров, кошелек может отказаться от услуг этого пира. В большинстве случаев единственным последствием неточного фильтра является увеличение расхода трафика кошелька по сравнению с ожидаемым.

Экономия трафика за счет кодирования с потерями

Данные блока по транзакциям, которые необходимо передать, – это скрипт выхода. Эти скрипты различаются по длине и соответствуют шаблонам, а значит, различия между ними не будут распределены одинаково, как хотелось бы. Однако во многих местах этой книги уже было показано, что с помощью хеш-функции можно создать обязательство для определенных данных, а также получить значение, похожее на случайно выбранное число.

В других разделах этой книги рассматривались криптографически защищенные хеш-функции, которые гарантируют стойкость обязательств и невозможность отличить результаты на выходе от случайных. Однако существуют более быстрые и настраиваемые хеш-функции без криптографии, например функция SipHash, которая используется для компактных фильтров блоков.

Подробности об используемом алгоритме описаны в BIP158, а суть в том, что каждый скрипт выхода сокращается до 64-битного обязательства с помощью SipHash и некоторых арифметических операций. Можно представить это как использование набора больших чисел и их сокращение до более коротких чисел – процесс, при котором теряются данные. Именно поэтому он называется кодированием с потерями (lossy encoding). Теряя часть информации, мы избавляемся от необходимости ее дальнейшего хранения, что позволяет сэкономить место. В данном случае обычный скрипт выхода длиной 160 бит и более сокращается до 64 бит.

Использование компактных фильтров блоков

Все 64-битные значения для каждого обязательства перед скриптом выхода в блоке сортируются, дубликаты удаляются, а GCS строится путем нахождения разницы (дельты) между каждой записью. Затем этот компактный фильтр блоков распространяется пирами среди их клиентов (например, кошельков).

Клиент с помощью дельты восстанавливает исходные обязательства. Клиент, например кошелек, также берет все скрипты выхода, которые он отслеживает, и генерирует обязательства аналогично BIP158. Он проверяет, совпадает ли любое из сгенерированных им обязательств с обязательствами в фильтре.

Вспомним пример о потерях компактных фильтров блоков, аналогичных сокращению числа. Представим, что клиент ищет блок с числом 123 456, а точный (но с потерями) компактный фильтр блоков содержит число 1234. Когда клиент увидит 1234, он загрузит соответствующий блок.

Существует 100%-ная гарантия того, что точный фильтр с записью 1234 позволит клиенту узнать о блоке с записью 123 456, и это называется *истинно положительным* (*true positive*) результатом. Однако существует вероятность того, что блок может содержать 123 400, 123 401 или почти сотню других записей, которые не являются целью поиска клиента (в данном примере), и это называется *ошибочно позитивным* (*false positive*) результатом, или просто ложным срабатыванием.

Стопроцентное истинно положительное совпадение – это замечательно. Это указывает на то, что кошелек имеет возможность положиться на компактные фильтры блоков в поисках каждой транзакции, затрагивающей этот кошелек. Ненулевая частота ложных срабатываний означает, что кошелек в итоге загрузит несколько блоков без интересных для него транзакций. Главным следствием этого станет использование клиентом дополнительного трафика, что нельзя назвать большой проблемой. Фактический процент ложных срабатываний компактных фильтров блоков BIP158 очень низок, поэтому особой роли это не играет. Коэффициент ложных срабатываний также может помочь улучшить конфиденциальность клиента, как это происходит с фильтрами Блума, хотя тем, кто стремится к максимальной конфиденциальности, все равно следует использовать свой собственный полный узел.

В долгосрочной перспективе некоторые разработчики выступают за то, чтобы блоки фиксировали фильтр для этого блока, а наиболее вероятная схема – чтобы каждая транзакция coinbase фиксировала фильтр для этого блока. Полные узлы будут сами вычислять фильтр для каждого блока и принимать блок только в том случае, если он содержит точное обязательство. Это позволило бы облегченному клиенту загрузить 80-байтовый заголовок блока, (как правило) небольшую транзакцию coinbase и фильтр для этого блока, чтобы получить убедительные доказательства корректности фильтра.

Облегченные клиенты и конфиденциальность

Облегченные клиенты обеспечивают более низкую конфиденциальность, чем полные узлы. Полный узел загружает все транзакции и поэтому не разглашает информацию об использовании того или иного адреса в своем кошельке. Об-

легченный клиент загружает только те транзакции, которые каким-то образом связаны с его кошельком.

Фильтры Блума и компактные фильтры блоков помогают уменьшить масштабы снижения конфиденциальности. Без них облегченному клиенту пришлось бы в явном виде перечислять интересующие его адреса, что привело бы к серьезному нарушению конфиденциальности. Но даже с фильтрами злоумышленник, следящий за трафиком облегченного клиента или подключенный к нему напрямую как узел в P2P-сети, со временем сможет собрать достаточное количество информации для выяснения адресов в кошельке облегченного клиента.

Соединения с шифрованием и аутентификацией

Большинство новых пользователей Биткойна считают, что сетевые сообщения узла Биткойна защищены шифрованием. На самом деле оригинальная реализация Биткойна передает данные полностью в открытом виде, как и современная реализация Bitcoin Core на момент написания книги.

Для повышения конфиденциальности и безопасности P2P-сети Биткойн есть решение, которое обеспечивает шифрование коммуникаций: сервис транспортировки Tor (Tor transport).

Tor (*The Onion Routing network*) – это программный проект и сеть, которая предлагает шифрование и инкапсуляцию данных с помощью произвольно выбираемых сетевых маршрутов, обеспечивающих анонимность, отсутствие возможности отслеживания и конфиденциальность.

В Bitcoin Core есть несколько вариантов конфигурации, которые позволяют запустить узел Биткойна с трафиком через сеть Tor. Кроме того, Bitcoin Core может предложить скрытый сервис Tor, который дает другим узлам Tor возможность подключаться к вашему узлу напрямую через Tor.

Начиная с Bitcoin Core версии 0.12 узлы будут предлагать скрытый сервис Tor автоматически, как только смогут подключиться к локальному сервису Tor. Если у вас установлен Tor и процесс Bitcoin Core запускается от имени пользователя с достаточными правами для доступа к файлу cookie аутентификации Tor, он будет работать автоматически. Используйте флаг `debug` для включения отладки Bitcoin Core в сервисе Tor следующим образом:

```
$ bitcoind --daemon --debug=tor
```

Вы должны увидеть в логах `tor: ADD_ONION successful`, что указывает на успешное добавление скрытого сервиса Bitcoin Core в сеть Tor.

Дополнительные инструкции по запуску Bitcoin Core как скрытого сервиса Tor можно найти в документации Bitcoin Core (*docs/tor.md*) и различных онлайн-учебниках.

Мемпулы и орфанные пулы

Почти каждый узел сети Биткойн ведет временный список неподтвержденных транзакций, который называют *пулом памяти*, или просто *мемпулом* (*mem-*

pool). Узлы используют его для отслеживания транзакций, которые уже известны сети, но еще не включены в блокчейн, – так называемых *неподтвержденных транзакций* (*unconfirmed transactions*).

По мере получения и верификации неподтвержденных транзакций они добавляются в мемпул и передаются соседним узлам для распространения по сети.

Некоторые реализации узлов также поддерживают отдельный пул так называемых орфанных («осиротевших», или «бесхозных») транзакций (orphaned transactions). Если данные входа транзакции относятся к еще неизвестной транзакции, например к отсутствующей родительской, эта орфанная транзакция будет временно храниться в так называемом орфанном пуле (orphan pool) до тех пор, пока не поступит родительская транзакция.

После добавления транзакции в мемпул производится проверка орфанного пула на наличие орфанных транзакций, которые ссылаются на выходы этой транзакции (дочерние транзакции). Затем проверяются все совпадающие орфанные транзакции. Если они корректны, то их удаляют из орфанного пула и добавляют в мемпул, завершая цепочку с родительской транзакцией. С учетом вновь добавленной транзакции, которая больше не является орфанной, процесс повторяется рекурсивно в поисках дальнейших потомков, пока их больше не найдется. Благодаря этому процессу появление родительской транзакции запускает каскадное восстановление всей цепочки взаимозависимых транзакций, воссоединяя «сирот» с их родителями по всей цепочке.

Некоторые реализации Биткойна также поддерживают базу данных UTXO, которая представляет собой набор всех неизрасходованных выходов в блокчейне. Она отличается от мемпула другим набором данных. В отличие от мемпула и орфанного пула, база данных UTXO содержит миллионы записей о неизрасходованных выходах транзакций – все неизрасходованное начиная с блока генезиса. База данных UTXO хранится в виде таблицы в постоянном хранилище.

В то время как мемпулы и орфанные пулы, отражающие локальную перспективу отдельного узла, могут значительно отличаться от узла к узлу в зависимости от времени запуска или перезапуска узла, база данных UTXO выражает сложившийся консенсус сети и по этой причине обычно не отличается на разных узлах.

Теперь, после ознакомления с многочисленными типами и структурами данных, используемых узлами и клиентами для передачи данных по сети Биткойн, пришло время рассмотреть программное обеспечение, которое отвечает за безопасность и работоспособность сети.

Глава 11

Блокчейн

Блокчейн – это история каждой подтвержденной транзакции Биткойна. Именно он позволяет каждому полному узлу независимо от других узлов определять ключи и скрипты, управляющие биткойнами. В этой главе будет рассмотрена структура блокчейна, а также использование криптографических обязательств и других ухищрений, благодаря которым все его части могут быть легко подтверждены полными узлами (и иногда облегченными клиентами).

Структура данных блокчейна представлена упорядоченным взаимосвязанным списком блоков транзакций. Блокчейн может храниться в виде обычного файла или в простейшей базе данных. Блоки связаны «задом наперед», каждый из них ссылается на предыдущий в цепочке. Блокчейн часто представляют в виде вертикальной стопки, где блоки наслаиваются друг на друга, а первый блок служит основой этой стопки. Визуализация блоков, уложенных друг на друга, приводит к использованию таких терминов, как «высота» (height) для обозначения расстояния от первого блока и «верх» (top) или «вершина» (tip) для обозначения последнего добавленного блока.

Каждый блок в блокчейне идентифицируется по хешу, сгенерированному с помощью криптографического хеш-алгоритма SHA256 в заголовке блока. Каждый блок также фиксирует связь с предыдущим блоком, называемым *родительским* (*parent*), через поле «хеш предыдущего блока» (previous block hash) в заголовке блока. Последовательность хешей, связывающих каждый блок с родительским, создает цепочку, ведущую к самому первому созданному блоку. Его называют *блоком генезиса* (*genesis block*).

Хотя у каждого блока есть только один родительский блок, он может иметь несколько дочерних блоков (потомков – children). Каждый из дочерних блоков фиксирует один и тот же родительский блок. Несколько дочерних блоков образуются во время «развилки» (fork) блокчейна – временной ситуации, которая возникает при обнаружении разных блоков почти одновременно разными майнерами (см. раздел «Сборка и выбор цепочек блоков»). В итоге только один дочерний блок попадает в блокчейн после признания всеми полными узлами, и «развилка» устраняется.

Поле «хеш предыдущего блока» расположено в заголовке самого блока и, соответственно, влияет на хеш текущего блока. Любое изменение родительского блока требует изменения хеша дочернего блока, что приводит к изменению указателя «внука», который, в свою очередь, изменяет своего потомка, и так далее. Такая последовательность гарантирует, что в случае, если за блоком следует множество поколений, его уже не удастся изменить без принудительного пересчета всех последующих блоков. Поскольку подобный пересчет потребовал бы огромных вычислительных ресурсов (и, следовательно, затрат энергии), существование длинной цепочки блоков не позволяет изменить глубокую историю блокчейна, что является ключевой характеристикой безопасности Биткойна.

Блокчейн можно сравнить со слоями геологической формации или образцом керна из ледника. Поверхностные слои могут меняться в зависимости от времени года или даже разрушаться, не успев толком осесть. Но стоит углубиться на несколько сантиметров, как геологические слои становятся все более и более стабильными. Уже на глубине нескольких десятков метров можно увидеть снимок прошлого, которое оставалось нетронутым в течение миллионов лет. В блокчейне последние несколько блоков могут быть изменены в случае реорганизации цепи из-за развилки. Шесть верхних блоков – это как несколько сантиметров верхнего слоя почвы. Но если углубиться в блокчейн более чем на шесть блоков, вероятность изменения блоков становится все меньше и меньше. После 100 блоков стабильность становится настолько высокой, что транзакция coinbase – транзакция, содержащая вознаграждение в биткойнах за создание нового блока, уже может быть потрачена. Хотя протокол всегда позволяет отменить цепочку с помощью более длинной цепочки и хотя возможность отмены любого блока всегда существует, вероятность такого события со временем уменьшается, пока не становится бесконечно малой.

Структура блока

Блок является контейнером данных, в котором хранятся транзакции для включения в блокчейн. Блок состоит из заголовка с метаданными, за которым следует длинный список транзакций, составляющих основную часть его объема. Заголовок блока занимает 80 байт, в то время как общий размер всех транзакций в блоке может достигать примерно 4 000 000 байт. Таким образом, полный блок со всеми транзакциями может быть почти в 50 000 раз больше, чем заголовок блока. В табл. 11.1 описано хранение структуры блока в Bitcoin Core.

Таблица 11.1. Структура блока

Размер	Поле	Описание
4 байта	Размер блока	Размер блока в байтах, следующего за этим полем
80 байт	Заголовок блока	Заголовок блока состоит из нескольких полей
1–3 байта (compactSize)	Счетчик транзакций	Количество последующих транзакций
Переменный	Транзакции	Транзакции, записанные в этом блоке

Заголовок блока

Заголовок блока включает метаданные этого блока, как показано в табл. 11.2.

Таблица 11.2. Структура заголовка блока

Размер	Поле	Описание
4 байта	Версия	Изначально поле версии; со временем его назначение изменилось
32 байта	Хеш предыдущего блока	Хеш предыдущего (родительского) блока в цепочке
32 байта	Корень дерева Меркла	Корневой хеш дерева Меркла для транзакций этого блока
4 байта	Временная метка	Приблизительное время создания этого блока (время эпохи Unix)
4 байта	Цель	Компактное кодирование целевого показателя доказательства работы для этого блока
4 байта	Нонс	Произвольные данные для алгоритма доказательства работы

Поля «нонс» (nonce), «цель» (target) и «временная метка» (timestamp) используются в процессе майнинга. Подробнее о них будет рассказано в главе 12.

Идентификаторы блока: хеш заголовка блока и высота блока

Основным идентификатором блока является его криптографический хеш, который определяется путем двойного хеширования заголовка блока по алгоритму SHA256. Полученный 32-байтовый хеш называется *хешем блока* (*block hash*), но более корректно его следует называть *хешем заголовка блока* (*block header hash*), поскольку для его вычисления используется только заголовок блока. Например, `000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f` – это хеш первого блока в блокчейне Bitcoin. Хеш блока идентифицирует блок единственным и однозначным образом. Он может быть получен независимо любым узлом посредством простого хеширования заголовка блока.

Отметим, что хеш блока на самом деле не входит в структуру данных блока. Хеш блока вычисляется каждым узлом при получении блока из сети. Хеш блока может храниться в отдельной таблице базы данных как часть метаданных блока для более быстрого индексирования и извлечения блоков из памяти компьютера.

Другим способом идентификации блока служит его положение в блокчейне, называемое *высотой блока* (*block height*). Блок генезиса находится на высоте 0 (ноль) и является тем же блоком, на который ранее ссылался следующий хеш блока `000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f`. Таким образом, блок можно идентифицировать двумя способами: по хешу блока или по высоте блока. Каждый последующий блок, добавленный «поверх» первого блока, располагается в блокчейне на одну позицию «выше», как коробки, поставленные одна на другую. Во время написания этой книги в середине 2023 года была достигнута высота блока 800 000. Это означает, что поверх первого блока, созданного в январе 2009 года, было добавлено 800 000 блоков.

Высота блока, в отличие от хеша блока, не является уникальным идентификатором. Хотя у отдельного блока всегда будет определенная и неизменная высота блока, обратное не верно – высота блока не всегда идентифицирует отдельный блок. Два или более блоков могут иметь одинаковую высоту блока, конкурируя за одну и ту же позицию в блокчейне. Этот сценарий подробно рассматривается в разделе «Сборка и выбор цепочек блоков». В ранних версиях блокчейна высота блока также не была частью структуры данных блока; она не хранилась внутри блока. Каждый узел динамически определял позицию (высоту) блока в блокчейне, когда получал его из сети Биткойн. Более позднее изменение протокола (BIP34) привело к включению высоты блока в транзакцию coinbase, хотя цель этого изменения заключалась в создании отдельной транзакции coinbase для каждого блока. Узлы по-прежнему должны динамически определять высоту блока для подтверждения поля coinbase. Высота блока также может храниться вместе с метаданными в индексированной таблице базы данных для более быстрого поиска.

Хеш блока всегда идентифицирует конкретный блок однозначным образом. Блок также всегда характеризуется определенной *высотой блока*. Однако далеко не всегда конкретная высота блока идентифицирует только один блок. Наоборот, два или более блоков могут конкурировать за одну позицию в блокчейне.

Блок генезиса

Первый блок в блокчейне называется *блоком генезиса* (*genesis block*). Он был создан в 2009 году. Этот блок является общим предком всех блоков в блокчейне. То есть если начать с любого блока и проследить цепочку в обратном направлении, в итоге можно прийти к блоку генезиса.

Каждый узел всегда начинает с блокчейна, состоящего минимум из одного блока, потому что блок генезиса закодирован в Bitcoin Core статически, и его невозможно изменить. Каждый узел всегда «знает» хеш и структуру блока генезиса, точное время его создания и даже единственную транзакцию в нем. Таким образом, у каждого узла есть исходная точка блокчейна, безопасный «корень», на основе которого можно построить надежный блокчейн.

Посмотреть статически закодированный блок генезиса можно в клиенте Bitcoin Core, в файле *chainpar-ams.cpp* (https://github.com/bitcoin/bitcoin/blob/3955c3940eff83518c186facfec6f50545b5aab5/src/chainparams.cpp#L123).

Блоку генезиса принадлежит следующий хеш идентификатора:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Практически на любом сайте-проводнике блоков, например blockstream.info, можно найти хеш этого блока, и всегда найдется страница, описывающая содержимое этого блока, с URL, содержащим этот хеш: https://blockstream.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f.

Кроме того, блок можно получить из Bitcoin Core с помощью командной строки:

```
$ bitcoin-cli getblock \
  000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

```
{
  "hash": "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "confirmations": 790496,
  "height": 0,
  "version": 1,
  "versionHex": "00000001",
  "merkleroot": "4a5e1e4baab89f3a32518a88c3[...]76673e2cc77ab2127b7afdeda33b",
  "time": 1231006505,
  "mediantime": 1231006505,
  "nonce": 2083236893,
  "bits": "1d00ffff",
  "difficulty": 1,
  "chainwork": "[...]0000000000000000000000000000000000000000000000000100010001",
  "nTx": 1,
  "nextblockhash": "00000000839a8e6886ab5951d7[...]fc90947ee320161bbf18eb6048",
  "strippedsize": 285,
  "size": 285,
  "weight": 1140,
  "tx": [
    "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
  ]
}
```


Блок генезиса содержит сообщение. Вход транзакции coinbase содержит текст «The Times 03/Jan/2009 Канцлер на пороге второго спасения банков» (The Times 03/Jan/2009 Chancellor on brink of second bailout for banks). Это сообщение служит доказательством наиболее ранней даты создания данного блока, ссылаясь на заголовок британской газеты The Times. Оно также служит шуточным напоминанием о значимости независимой денежной системы, поскольку запуск биткойна произошел одновременно с беспрецедентным мировым валютным кризисом. Это послание добавил в первый блок Сатоши Накамото (Satoshi Nakamoto), создатель Биткойна.

Взаимосвязь блоков в блокчейне

Полные узлы Биткойна проверяют каждый блок в блокчейне после генезисного блока. Их локальное представление блокчейна постоянно обновляется по мере нахождения новых блоков и их дальнейшего использования для продления цепочки. Когда узел получает входящие блоки из сети, он проводит их валидацию, а затем связывает со своим вариантом существующего блокчейна. Для установления связи узел изучает заголовок входящего блока и ищет «хеш предыдущего блока».

Предположим, что у узла есть 277 314 блоков в локальной копии блокчейна. Последний блок, о котором знает узел, – это блок 277 314 со следующим хешем заголовка блока:

```
00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249
```

Затем узел Биткойна получает из сети новый блок, который он анализирует следующим образом:


```
{
  "size" : 43560,
```

```
  "version" : 2,
  "previousblockhash" : "0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
  "merkleroot" : "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
  "time" : 1388185038,
  "difficulty" : 1180923195.25802612,
  "nonce" : 4215469401,
  "tx" : [ "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
           "[... many more transactions omitted ...]",
           "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
  ]
}
```

Изучая этот новый блок, узел находит поле `previousblockhash`, которое содержит хеш его родительского блока. Это известный узлу хеш последнего блока в цепочке на высоте 277 314. Таким образом, этот новый блок является дочерним по отношению к последнему блоку в цепочке и продолжает существующую цепочку. Узел добавляет этот новый блок в конец цепочки, делая блокчейн длиннее с новой высотой 277 315. На рис. 11.1 показана цепочка из трех блоков, связанных ссылками в поле `previousblockhash`.

Деревья Меркла

Каждый блок в блокчейне Биткойна содержит сводную информацию обо всех транзакциях в этом блоке, для чего используется *дерево Меркла* (*merkle tree*).

Дерево Меркла, также известное как *двоичное хеш-дерево* (*binary hash tree*), является структурой данных, которая используется для эффективного анализа и верификации целостности больших массивов данных. Термин «дерево» употребляется в информатике для описания ветвящейся структуры данных, однако обычно такие деревья отображаются в перевернутом виде: «корень» вверху, а «листья» внизу диаграммы, как будет показано в следующих примерах.

Деревья Меркла используются в Биткойне для группировки всех транзакций в блоке, что позволяет получить общее обязательство по всему множеству транзакций и дает возможность эффективно проверять принадлежность транзакции к данному блоку. Дерево Меркла строится методом рекурсивного хеширования пар элементов до тех пор, пока не останется только один хеш, называемый *корнем* (*root*), или *корнем дерева Меркла* (*merkle root*). Криптографический хеш-алгоритм, используемый в деревьях Меркла Биткойна, – это дважды примененный SHA256, также называемый double-SHA256.

Если хешировать и объединить в дерево Меркла N элементов данных, то можно проверить, включен ли в него какой-либо элемент данных, выполнив примерно $\log_2(N)$ вычислений, что обеспечивает высокую эффективность этой структуры данных.

Дерево Меркла строится снизу вверх. В следующем примере для начала используются четыре транзакции, A, B, C и D, которые образуют листья дерева Меркла, как показано на рис. 11.2. Транзакции не хранятся в дереве Меркла; наоборот, их данные хешируются, и полученный хеш хранится в каждом узле листа как H_A, H_B, H_C и H_D:

```
H_A = SHA256(SHA256(Transaction A))
```

Далее последовательные пары узлов листьев суммируются в родительском узле путем объединения двух хешей и их совместного хеширования. Например, чтобы построить родительский узел H_{AB}, два 32-байтовых хеша дочерних узлов объединяются для создания 64-байтовой строки. Затем эта строка дважды хешируется для получения хеша родительского узла:

$H_{AB} = SHA256(SHA256(H_A || H_B))$

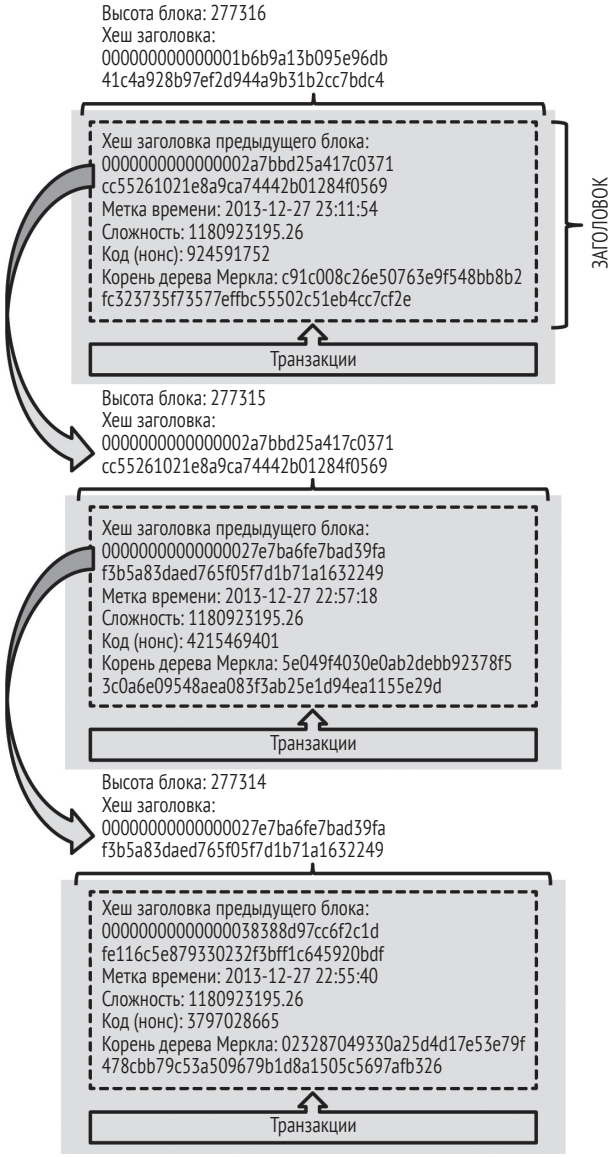


Рис. 11.1 ❖ Блоки в цепочке, связанные ссылками на хеш заголовка предыдущего блока

Процесс продолжается до того момента, пока на вершине не останется только один узел – так называемый корень Меркла. Этот 32-байтовый хеш хранится в заголовке блока и обобщает все данные во всех четырех транзакциях. На рис. 11.2 показан процесс вычисления корня путем попарного хеширования узлов.

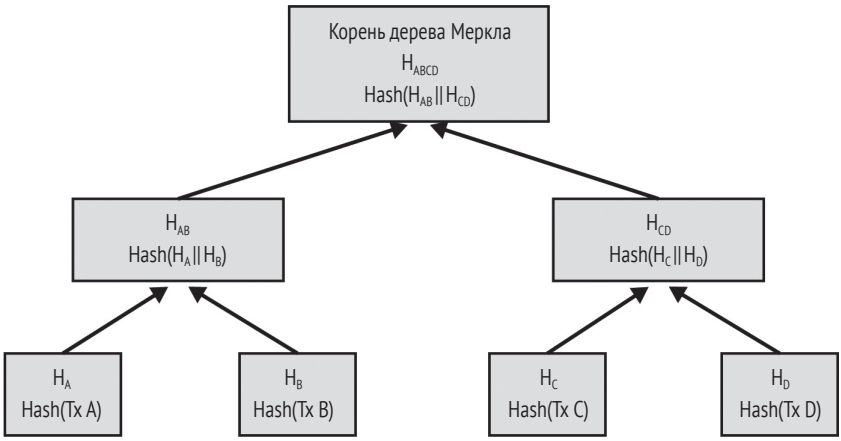


Рис. 11.2 ❖ Вычисление узлов в дереве Меркла

Поскольку дерево Меркла является бинарным деревом, ему необходимо четное количество узлов листьев. При нечетном количестве транзакций, которые необходимо объединить, хеш последней транзакции будет продублирован для создания четного количества узлов листьев. Это также называют сбалансированным деревом (balanced tree). Как показано на рис. 11.3, транзакция C дублируется. Подобным образом, если на каком-либо уровне необходимо обработать нечетное количество хешей, последний хеш дублируется.

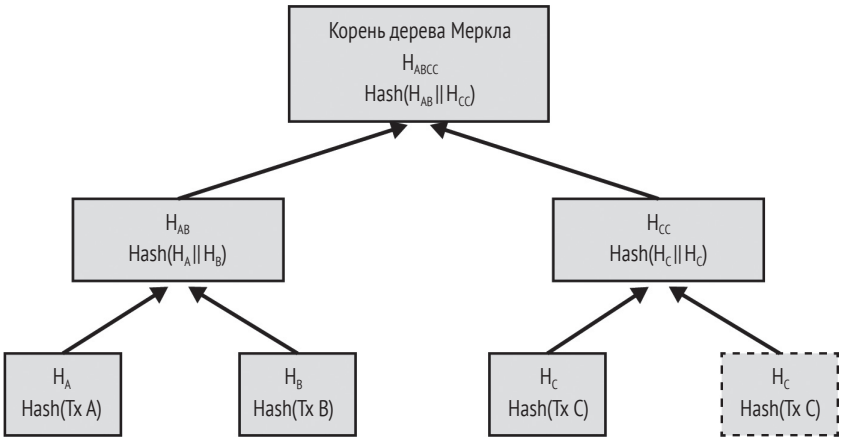


Рис. 11.3 ❖ Дублирование элемента данных для получения четного числа элементов данных

Недостаток концепции дерева Меркла в Биткойне

Расширенный комментарий в исходном коде Bitcoin Core, воспроизведенный здесь с небольшими изменениями, описывает существенную проблему в концепции дублирования нечетных элементов в дереве Меркла в Биткойне:

> ВНИМАНИЕ! Если вы читаете это, потому что изучаете криптовалюты и/или разрабатываете новую систему, в которой будут использоваться деревья Меркла, имейте в виду, что следующий алгоритм дерева Меркла имеет серьезный недостаток, связанный с дублированием txids, что приводит к уязвимости (CVE-2012-2459). Дело в том, что если количество хешей в списке на данном уровне нечетное, то последний из них дублируется перед вычислением следующего уровня (что необычно для деревьев Меркла). Это приводит к тому, что некоторые последовательности операций ведут к одному и тому же корню Меркла. Например, два дерева на рис. 11.4:

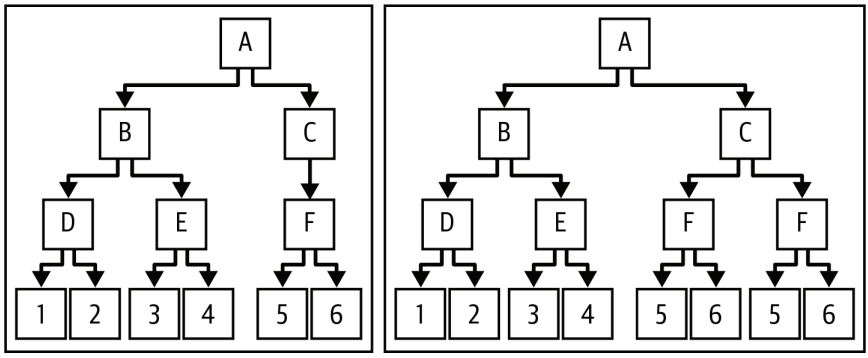


Рис. 11.4 ❖ Два дерева Меркла в формате Биткойна с одинаковым корнем, но разным количеством листьев

> Списки транзакций [1,2,3,4,5,6] и [1,2,3,4,5,6,5,6] (где 5 и 6 повторяются) приводят к одному и тому же корневому хешу A (потому что хеш обоих (F) и (F,F) равен C). Уязвимость возникает из-за возможности отправить блок с таким списком транзакций, с тем же корнем Меркла и тем же хешем блока, что и оригинальный, без дублирования, что приводит к ошибке при валидации. Однако если принимающий узел пометит этот блок как постоянно недействительный, он не сможет принять последующие немодифицированные (и, следовательно, потенциально действительные) версии того же блока. Для защиты от этого можно выявить ситуацию, когда два одинаковых хеша в конце списка будут хешированы вместе, и отнестись к этому так же, как к блоку с недействительным корнем Меркла. При условии отсутствия конфликтов с двойным SHA256 это позволит обнаружить все известные способы изменения транзакций без влияния на корень Меркла.
>
> – Взято из Bitcoin Core (*src/consensus/merkle.cpp*)

Этот метод построения дерева из четырех транзакций можно применять для построения деревьев любого размера. В Биткойне обычно насчитывается несколько тысяч транзакций в одном блоке, и все они сводятся точно таким же

образом, создавая всего 32 байта данных в виде единственного корня Меркла. На рис. 11.5 показано дерево, построенное из 16 транзакций. Обратите внимание, что хотя корень выглядит крупнее, чем узлы листьев на рисунке, его размер точно такой же – всего 32 байта. Неважно, одна транзакция или десять тысяч транзакций в блоке, корень Меркла всегда суммирует их в 32 байта.

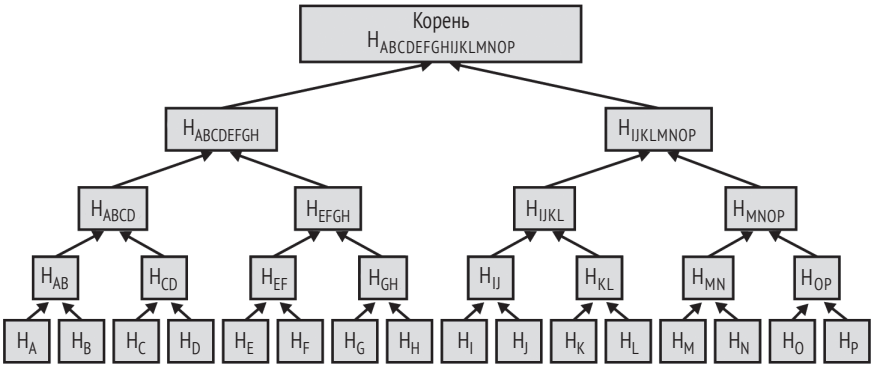


Рис. 11.5 ❖ Дерево Меркла, суммирующее множество элементов данных

Для доказательства включения конкретной транзакции в блок узел должен произвести примерно $\log_2(N)$ 32-байтовых хешей, образующих *путь аутентификации* (*authentication path*), или *путь Меркла* (*merkle path*), который связывает эту транзакцию с корнем дерева. Особенно это важно при увеличении числа транзакций, поскольку логарифм числа транзакций по основанию 2 растет гораздо медленнее. Это позволяет узлам Биткойна эффективно формировать пути из 10 или 12 хешей (320–384 байта), которые могут служить доказательством одной транзакции из более чем тысячи транзакций в блоке размером в несколько мегабайтов.

На рис. 11.6 показано, что узел может подтвердить включение транзакции K в блок, создав путь Меркла длиной всего в четыре 32-байтных хеша (всего 128 байт). Путь состоит из четырех хешей (показаны на заштрихованном фоне) H_L, H_{IJ}, H_{MNOP} и $H_{ABCDEFGH}$. Используя эти четыре хеша в качестве пути аутентификации, любой узел может доказать, что H_K (с черным фоном в нижней части диаграммы) включен в корень дерева Меркла путем вычисления четырех дополнительных парных хешей H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$ и корня дерева Меркла (обведен пунктирной линией).

Эффективность деревьев Меркла становится очевидной при увеличении масштаба. Самый большой из возможных блоков может содержать почти 16 000 транзакций объемом 4 000 000 байт, но для доказательства принадлежности любой из этих 16 000 транзакций к данному блоку требуется только копия транзакции, копия 80-байтового заголовка блока и 448 байт для доказательства Меркла. Таким образом, самое большое допустимое доказательство почти в 10 000 раз меньше самого большого из возможных блоков Биткойна.

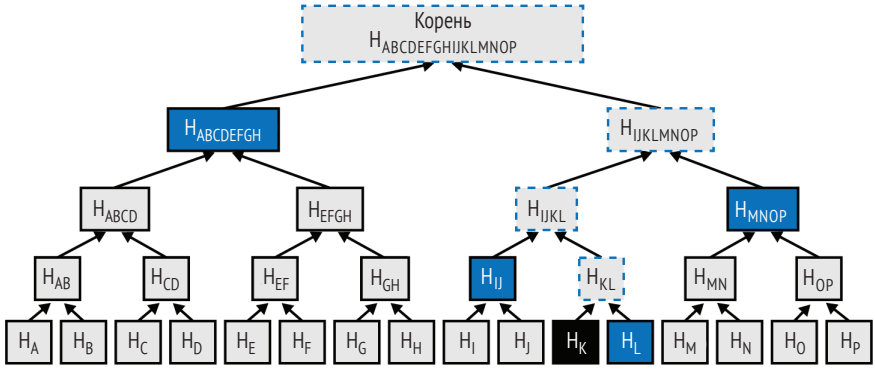


Рис. 11.6 ❖ Путь Меркла для доказательства включения элемента данных

Деревья Меркла и облегченные клиенты

Деревья Меркла широко используются облегченными клиентами. У них нет всех транзакций, и они не загружают полные блоки, а только их заголовки. Чтобы убедиться в наличии транзакции в блоке без необходимости их загрузки, используется путь Меркла.

В качестве примера рассмотрим облегченного клиента, которого интересуют входящие платежи на адрес в его кошельке. Облегченный клиент установит фильтр Блума (см. раздел «Фильтры Блума») на своих соединениях с пирами, для того чтобы ограничить количество получаемых транзакций только содержащими интересующие его адреса. Когда пир увидит транзакцию, соответствующую фильтру Блума, он отправит этот блок с помощью сообщения `merkleblock`. Сообщение `merkleblock` содержит заголовок блока, а также путь Меркла, который связывает интересующую транзакцию с корнем Меркла в блоке. Облегченный клиент может использовать этот путь Меркла для связи транзакции с заголовком блока и проверки принадлежности транзакции к блоку. Облегченный клиент также использует заголовок блока для привязки блока к остальным частям блокчейна. Сочетание этих двух связей – между транзакцией и блоком и между блоком и блокчейном – подтверждает запись транзакции в блокчейне. В итоге облегченный клиент получит менее килобайта данных для заголовка блока и пути Меркла, что более чем в тысячу раз меньше, чем для полного блока (около 2 Мбайт на момент написания книги).

Тестовые блокчейны Биткойна

Возможно, вы удивитесь, когда узнаете, что в Биткойне используется не один блокчейн, а несколько. «Главный» блокчейн Биткойна, созданный Сатоши Накамото 3 января 2009 года и содержащий блок генезиса, который описан в этой главе, называется mainnet. Существуют и другие блокчейны Биткойна, которые

используются для тестирования: на момент написания книги это *testnet*, *signet* и *regtest*. Рассмотрим каждый из них по очереди.

Testnet: площадка для тестирования Биткойна

Testnet – это название тестового блокчейна, сети и валюты, которые используются для испытаний. Сеть testnet представляет собой полнофункциональную действующую P2P-сеть с кошельками, тестовыми биткойнами (testnet coins), майнингом и всеми остальными функциями mainnet. Самое главное отличие заключается в том, что монеты testnet не имеют ценности.

Любые программные разработки для основной сети Биткойн можно предварительно опробовать в сети testnet с помощью тестовых монет. Так разработчики могут избежать финансовых потерь из-за ошибок, а сеть получает защиту от нежелательного воздействия ошибок.

Действующая сеть testnet называется testnet3. Это уже третья итерация сети testnet, перезапущенная в феврале 2011 года с целью снижения уровня сложности по сравнению с предыдущей версией. Testnet3 – это крупный блокчейн, объем которого в 2023 году достигнет 30 ГБайт. Потребуется время для его полной синхронизации и загрузки ресурсов на вашем компьютере. Не так много, как в mainnet, но и «облегченным» его тоже не назвать.

Testnet и другие тестовые блокчейны, описанные в этой книге, не используют те же префиксы адресов, что и в основной сети mainnet, для предотвращения случайной отправки реальных биткойнов на тестовый адрес. Адреса в основной сети mainnet начинаются с `1`, `3` или `bc1`. Адреса тестовых сетей, упомянутых в этой книге, начинаются с `m`, `n` или `tb1`. Другие тестовые сети или новые протоколы, разрабатываемые в тестовых сетях, могут использовать другие префиксы адресов или их модификации.

Использование сети testnet

Как и многие другие программы для Биткойна, Bitcoin Core полностью поддерживает работу в testnet в качестве альтернативы mainnet. В testnet работают все функции Bitcoin Core, включая кошелек, майнинг монет testnet и синхронизацию полного узла testnet.

Для запуска Bitcoin Core в testnet вместо mainnet нужно использовать специальный ключ `testnet`:

```
$ bitcoind -testnet
```

В логах будет видно, что bitcoind создает новый блокчейн в подкаталоге `testnet3` стандартного каталога bitcoind:

```
bitcoind: Using data directory /home/username/.bitcoin/testnet3
```

Для подключения к bitcoind используется инструмент командной строки `bitcoin-cli`, но его также необходимо переключить в режим testnet:

```
$ bitcoin-cli -testnet getblockchaininfo
{
  "chain": "test",
```

```
"blocks": 1088,
"headers": 139999,
"bestblockhash": "0000000063d29909d475a1c[...]368e56cce5d925097bf3a2084370128",
"difficulty": 1,
"mediantime": 1337966158,
"verificationprogress": 0.001644065914099759,
"chainwork": "[...]0000000000000000000000000000000000000000000044104410441",
"pruned": false,
"softforks": [

[...]
```

В testnet3 также можно запускать другие реализации для полных узлов, такие как `btcd` (написанный на Go) и `bcoin` (написанный на JavaScript) для экспериментов и обучения на других языках программирования и фреймворках.

В testnet3 поддерживаются все функции mainnet, включая segregated witness v0 и v1 (см. разделы «Сегрегированный свидетель (Segregated Witness)» и «Главный корень (Taproot)»). Поэтому testnet3 может использоваться и для тестирования функций сегрегированного свидетеля.

Проблемы с testnet

В testnet используются не только все структуры данных, как в Биткойне, но и почти такой же механизм безопасности доказательства работы. Существенные отличия testnet заключаются в минимальной сложности, вдвое меньшей, чем у Биткойна, и в том, что разрешается включать блок с минимальной сложностью, если временная метка этого блока прошла более чем через 20 минут после предыдущего блока.

К сожалению, механизм безопасности доказательства работы был разработан в расчете на экономические стимулы Биткойна, которых нет в тестовом блокчейне, поскольку ему по определению запрещено иметь ценность. В тестовом блокчейне транзакции содержат нечто, называемое комиссией, но эта комиссия не имеет никакой ценности. Это означает, что единственным стимулом для майнера testnet для включения транзакций является желание помочь пользователям и разработчикам в тестировании их программного обеспечения.

Увы, любители устраивать беспорядки в работе систем часто испытывают более сильный стимул, по крайней мере в краткосрочной перспективе. Поскольку майнинг доказательства работы создан по принципу отсутствия разрешений, майнить может кто угодно, независимо от его намерений. Это означает, что майнеры, нарушающие работу системы, могут создавать в testnet множество блоков подряд, не включая в них никаких пользовательских транзакций. Когда происходят такие атаки, testnet становится непригодным для пользователей и разработчиков.

Signet: доказательство полномочий сети testnet

Не существует известного способа для системы, зависящей от доказательства работы без разрешения, обеспечить высокоэффективный блокчейн без исполь-

зования экономических стимулов. Поэтому разработчики протокола Биткойна обратились к альтернативам. Главной целью было сохранение как можно большей части структуры Биткойна, чтобы программы могли работать в тестовой сети с минимальными изменениями и при этом обеспечить приемлемую для использования среду. Вторичной целью было создание структуры многократного использования, которая позволила бы разработчикам новых программ без особых усилий создавать свои собственные тестовые сети.

Решение, реализованное в Bitcoin Core и других программах, называется signet, как определено в BIP325. Signet – это тестовая сеть, в которой каждый блок должен содержать доказательство (например, подпись) того, что создание этого блока было санкционировано надежным источником.

Если майнинг в Биткойне носит безразрешительный характер – им может заниматься любой желающий, то майнинг в signet строго регламентирован. Заниматься им могут только обладатели разрешения. Такое изменение было бы совершенно неприемлемым для основной сети Биткойн – никто не стал бы использовать эту программу, но это вполне оправдано для тестовой сети, где монеты не имеют никакой ценности и единственной целью является тестирование программ и систем.

Сети BIP325 signet разработаны для облегчения создания собственных. Если вас не устраивает работа чужой сети signet, вы можете создать свою собственную и подключить к ней свое программное обеспечение.

Стандартная и пользовательские сети signet

В Bitcoin Core поддержка signet производится по умолчанию, и на момент написания книги, по мнению авторов, эта тестовая сеть была наиболее распространенной. Сейчас им управляют два участника этого проекта. Если запустить Bitcoin Core с параметром `signet` и без каких-либо других параметров, связанных с signet, использоваться будет именно он.

На момент написания этой книги стандартный (по умолчанию – default) signet насчитывал около 150 000 блоков и имел размер порядка гигабайта. Он поддерживает все те же функции, что и основная сеть Биткойна, но при этом также используется для тестирования предлагаемых обновлений в рамках проекта Bitcoin Inquisition, который представляет собой софт-форк Bitcoin Core, предназначенный только для работы с signet.

Если требуется использовать другой signet, так называемый пользовательский (кастомизированный – *custom*) signet, необходимо иметь представление о скрипте, который используется для определения авторизации блока и называется *скриптом запроса* (*challenge script*). Это стандартный скрипт Биткойна, поэтому он поддерживает такие функции, как мультиподпись для авторизации блоков несколькими людьми. Также может понадобиться подключение к seed-узлу, который предоставит адреса пиров на пользовательской сети signet. Например:

```
bitcoind -signet -signetchallenge=0123...cdef -signetseednode=example.com:1234
```

В настоящее время авторы рекомендуют проводить открытое тестирование программного обеспечения для майнинга на testnet3, а все остальное открытое тестирование программ для работы с Биткойном – в стандартной сети signet.

Для взаимодействия с выбранной вами сетью можно использовать параметр `-signet` вместе с `bitcoin-cli`, аналогично тому, как это делается в testnet. Например:

```
$ bitcoin-cli -signet getblockchaininfo
{
  "chain": "signet",
  "blocks": 143619,
  "headers": 143619,
  "bestblockhash": "000000c46cb3505ddd296537[...]ad1c5768e2908439382447572a93",
  "difficulty": 0.003020638517858618,
  "time": 1684530244,
  "mediantime": 1684526116,
  "verificationprogress": 0.999997961940662,
  "initialblockdownload": false,
  "chainwork": "[...]0000000000000000000000000000000000000000000019ab37d2194",
  "size_on_disk": 769525915,
  "pruned": false,
  "warnings": ""
}
```

Regtest: локальный блокчейн

Термин Regtest расшифровывается как «регрессивное тестирование» (Regression Testing). Это функция Bitcoin Core, которая позволяет создавать локальный блокчейн для тестирования. В отличие от signet и testnet3, которые являются открытыми и совместно используемыми тестовыми блокчейнами, блокчейн regtest предназначен для запуска в виде закрытой системы для локального тестирования. Блокчейн regtest запускается с нуля. Можно добавить в сеть другие узлы или запустить ее с одним узлом только для тестирования программы Bitcoin Core.

Для запуска Bitcoin Core в режиме regtest используется флаг `regtest`:

```
$ bitcoind -regtest
```

Как и в случае с testnet, Bitcoin Core инициализирует новый блокчейн в подкаталоге *regtest* вашего стандартного каталога bitcoind:

```
bitcoind: Using data directory /home/username/.bitcoin/regtest
```

Для использования средства командной строки необходимо также указать флаг `regtest`. Для проверки блокчейна regtest можно воспользоваться командой `getblockchaininfo`:

```
$ bitcoin-cli -regtest getblockchaininfo
{
```

```
  "chain": "regtest",
  "blocks": 0,
  "headers": 0,
  "bestblockhash": "0f9188f13cb7b2c71f2a335e3[...]b436012afca590b1a11466e2206",
  "difficulty": 4.656542373906925e-10,
  "mediantime": 1296688602,
  "verificationprogress": 1,
  "chainwork": "[...]0000000000000000000000000000000000000000000000000000000000000002",
  "pruned": false,
  [...]
```

Как можно видеть, блоков еще нет. Нужно создать кошелек по умолчанию, получить адрес, а затем добыть немного (500 блоков) для получения вознаграждения:

```
$ bitcoin-cli -regtest createwallet ""
$ bitcoin-cli -regtest getnewaddress
bcrt1qwvfhw8pf79kw6tvpmtxyxwcfnd2t4e8v6qfv4a
$ bitcoin-cli -regtest generatetoaddress 500 \
  bcrt1qwvfhw8pf79kw6tvpmtxyxwcfnd2t4e8v6qfv4a
[
  "3153518205e4630d2800a4cb65b9d2691ac68eea99afa7fd36289cb266b9c2c0",
  "621330dd5bdabcc03582b0e49993702a8d4c41df60f729cc81d94b6e3a5b1556",
  "32d3d83538ba128be3ba7f9dbb8d1ef03e1b536f65e8701893f70dcc1fe2dbf2",
  ...,
  "32d55180d010ffebabf1c3231e1666e9eeed02c905195f2568c987c2751623c7"
]
```

На майнинг всех этих блоков уйдет всего несколько секунд, что, безусловно, облегчает тестирование. Если проверить баланс кошелька, то можно увидеть, что за первые 400 блоков было начислено вознаграждение (для того чтобы потратить вознаграждение coinbase, его глубина должна составлять 100 блоков):

```
$ bitcoin-cli -regtest getbalance
12462.50000000
```

Использование тестовых блокчейнов для разработки

Различные блокчейны Биткойна (regtest, signet, testnet3, mainnet) предлагают широкий спектр тестовых сред для разработки Биткойна. Можно использовать тестовые блокчейны для разработки Bitcoin Core или другого клиента полноузлового консенсуса; для разработки приложений, таких как кошелек, биржа, сайт электронной коммерции; или даже для разработки новых смарт-контрактов и сложных скриптов.

Тестовые блокчейны можно использовать для создания конвейера разработки. По мере создания кода его можно тестировать локально в regtest. Когда будете готовы опробовать его в открытой сети, переключитесь на signet или testnet, чтобы опробовать свой код в более динамичной среде с большим раз-

нообразием приложений. Наконец, когда вы убедитесь в работоспособности кода, можно переключиться на mainnet, чтобы развернуть его в полноценном режиме. По мере внесения изменений, улучшений, исправления ошибок и т. д. запускайте конвейер снова, развертывая каждое изменение сначала в regtest, затем в signet или testnet и, наконец, в полноценном режиме.

Теперь, когда мы узнали о содержимом блокчейна и о том, как криптографические обязательства надежно связывают его различные части воедино, перейдем к рассмотрению специальных обязательств, которые обеспечивают безопасность вычислений и гарантируют, что ни один блок не может быть изменен без признания недействительными всех остальных блоков, построенных на его основе. Речь – о функции майнинга Биткойна.

Глава 12

Майнинг и консенсус

Слово «майнинг» («добыча» – mining) в некоторой степени вводит в заблуждение. Хотя майнинг мотивируется вознаграждением, основной его целью является не сама премия или генерация новых биткойнов. Если рассматривать майнинг лишь как процесс получения биткойнов, можно ошибочно полагать, что средства (стимулы) являются целью процесса. Майнинг – это процесс, лежащий в основе децентрализованного расчетного механизма, с помощью которого происходят валидация и проверка транзакций. Майнинг – одно из тех изобретений, которые делают Биткойн особенным, – децентрализованный механизм консенсуса, лежащий в основе цифровой наличности P2P.

Майнинг *защищает систему Биткойн* и позволяет достичь консенсуса в масштабах сети без участия централизованной структуры. Вознаграждение в виде вновь добытых биткойнов и комиссии за транзакции – это система стимулов, которая обеспечивает согласованность действий майнеров с вопросами безопасности сети и одновременной реализацией денежных потоков.

Майнинг – это один из механизмов *децентрализованного обеспечения безопасности консенсуса* в Биткойне.

Майнеры записывают новые транзакции в глобальный блокчейн. Новый блок с транзакциями, осуществленными с момента появления последнего блока, *майнится* в среднем каждые 10 минут, в результате чего эти транзакции добавляются в блокчейн. Транзакции, ставшие частью блока и добавленные в блокчейн, считаются *подтвержденными* (*confirmed*). Это дает новым владельцам биткойнов возможность убедиться в надежности защиты биткойнов, которые они получили при совершении этих транзакций.

Кроме того, транзакции в блокчейне располагаются в *топологическом порядке*, который определяется их положением в блокчейне. Одна транзакция считается более ранней, чем другая, если она появилась в более раннем блоке или если она появилась раньше в том же блоке. В протоколе Биткойна транзакция действительна только в том случае, если она расходует выходы транзакций, появившихся в блокчейне раньше (в том же блоке или в более раннем блоке), и только если ни одна из предыдущих транзакций не расходовала эти же выходы. В пределах одной цепочки блоков соблюдение топологического порядка

гарантирует, что две действительные транзакции не смогут потратить один и тот же выход, и это устраняет проблему *двойных расходов* (*double spending*).

В некоторых протоколах на основе Биткойна топологический порядок его транзакций также используется для установления последовательности событий. Подробнее об этой идее говорится в разделе «Одноразовые печати».

В обмен на обеспечиваемую майнерами безопасность они получают два вида вознаграждения: новые биткойны, создаваемые с каждым новым блоком (так называемая *субсидия – subsidy*), и комиссии за все входящие в блок транзакции. Чтобы заработать это вознаграждение, майнеры соревнуются в решении задачи на основе криптографического хеш-алгоритма. Решение задачи, называемое доказательством работы (proof of work), включается в новый блок и служит доказательством затраченных майнером значительных вычислительных усилий. Соревнование в решении алгоритма доказательства работы для получения вознаграждения и права записывать транзакции в блокчейн лежит в основе модели безопасности Биткойна.

Денежная масса Биткойна создается в процессе, аналогичном выпуску центральным банком новых денег посредством печати банкнот. Максимальное количество вновь созданных биткойнов, которое майнер может добавить в блок, уменьшается примерно каждые четыре года (точнее, каждые 210 000 блоков). В январе 2009 года она составляла 50 биткойнов за блок, а в ноябре 2012 года уменьшилась вдвое (halved), до 25 биткойнов за блок. В июле 2016 года она снова уменьшилась вдвое, до 12,5 биткойна, а в мае 2020 года – до 6,25. Согласно этой формуле, вознаграждение за майнинг уменьшается по экспоненте примерно до 2140 года, когда будут выпущены все биткойны. После 2140 года новые биткойны выпускаться не будут.

Майнеры биткойнов также зарабатывают на транзакциях. Каждая транзакция может включать комиссию за транзакцию в виде избытка биткойнов между входом и выходом транзакции. Победивший майнер биткойнов получает «сдачу» (change) с транзакций, включенных в победивший блок. В настоящее время вознаграждение обычно составляет лишь небольшой процент от дохода майнера, а основная часть поступает от вновь добытых биткойнов. Однако со временем вознаграждение уменьшается, а количество транзакций в блоке увеличивается, и все большая часть дохода от майнинга будет приходиться на комиссию.

Со временем в вознаграждении за майнинг станет преобладать комиссия за транзакции, и она будет основным стимулом для майнеров. После 2140 года количество новых биткойнов в каждом блоке упадет до нуля, и майнинг будет стимулироваться только комиссией за транзакции. В этой главе вначале будет рассмотрен майнинг как механизм денежного обеспечения, а затем будет проанализирована самая важная функция майнинга: децентрализованный механизм консенсуса, лежащий в основе безопасности Биткойна.

Чтобы понять суть майнинга и консенсуса, нужно проследить за транзакцией Алисы в момент ее получения и добавления в блок на майнинговом оборудовании Jing. Затем мы будем следить за процессом майнинга блока, его добавлением в блокчейн и принятием сетью Биткойн в процессе достижения возникающего консенсуса.

Экономика Биткойна и создание денежных средств

Биткойны майнятся в процессе создания каждого блока с фиксированной и убывающей со временем периодичностью. Каждый блок, генерируемый в среднем каждые 10 минут, содержит совершенно новые биткойны, созданные из ничего. Каждые 210 000 блоков, или примерно раз в четыре года, скорость эмиссии уменьшается на 50 %. По итогам первых четырех лет работы сети каждый блок содержал 50 новых биткойнов.

Первое «уполовинивание» (так называемый «*халвинг*», *halving*) произошло на блоке 210 000. Следующий ожидаемый халвинг после публикации этой книги произойдет на блоке 840 000, который, вероятно, будет выпущен в апреле или мае 2024 года. Курс новых биткойнов будет экспоненциально уменьшаться на протяжении 32 таких халвингов до блока 6 720 000 (майнинг ожидается примерно в 2137 году), когда он достигнет минимальной денежной единицы в 1 сатоши. Наконец, после 6,93 млн блоков, примерно в 2140 году, будет выпущено почти 2 099 999 997 690 000 сатоши, или почти 21 млн биткойнов. После этого в блоках не будет новых биткойнов, а майнеры будут получать вознаграждение исключительно за счет комиссии по транзакциям. На рис. 12.1 показано общее количество биткойнов в обращении с течением времени, по мере уменьшения эмиссии этой денежной единицы.

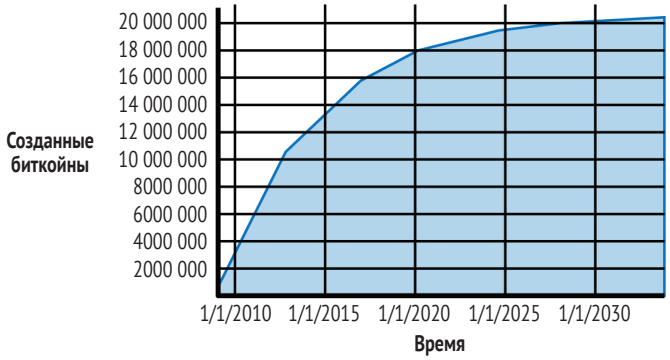


Рис. 12.1 ❖ Объем предложения биткойнов с течением времени с учетом геометрически убывающей эмиссии

Максимальное количество добытых биткойнов – это верхний предел возможного вознаграждения за их добычу. На практике майнер может намеренно добыть блок с меньшей суммой, чем полное вознаграждение. Такие блоки уже были добыты, и в будущем их может быть добыто еще больше, что приведет к снижению общей эмиссии этой денежной единицы.

В коде, приведенном в примере 12.1, вычисляется общее количество биткойнов, которое планируется выпустить.

Пример 12.1 ❖ Скрипт для расчета общего объема эмиссии биткойнов

```
# Первоначальное вознаграждение за блок для майнеров составляло 50 BTC
start_block_reward = 50
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000


def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
    while current_reward > 0:
        total += reward_interval * current_reward
        current_reward /= 2
    return total

print("Total BTC to ever be created:", max_money(), "Satoshis")
```

В примере 12.2 показан вывод с помощью этого скрипта.

Пример 12.2 ❖ Запуск скрипта max_money.py

```
$ python max_money.py
Total BTC to ever be created: 2099999997690000 Satoshis
```

Конечный и уменьшающийся объем эмиссии обеспечивает фиксированную денежную массу, которая противостоит инфляции. В отличие от фиатной валюты, которая может быть напечатана центральным банком в бесконечном количестве, в Биткойне ни одна сторона не может раздуть предложение.

Дефляционные деньги

Наиболее важным и обсуждаемым следствием фиксированной и уменьшающейся денежной эмиссии является стремление валют к *дефляции*. Дефляция – это эффект повышения стоимости из-за несоответствия спроса и предложения, что приводит к росту стоимости (и обменного курса) валюты. Ценовая дефляция противоположна инфляции; она означает, что деньги с течением времени приобретают большую покупательную способность.

Многие экономисты считают дефляционную экономику катастрофой, которой следует избегать любой ценой. Это связано с тем, что в период быстрой дефляции люди склонны хранить деньги, а не тратить их, в ожидании снижения цен. Подобное явление наблюдалось в Японии в «потерянное десятилетие», когда полный крах спроса загнал валюту в дефляционную спираль.

Эксперты по Биткойну утверждают, что дефляция сама по себе не является чем-то плохим. Скорее, дефляция ассоциируется с крахом спроса, поскольку это самый очевидный из доступных для исследования примеров дефляции. В фиатной валюте с возможностью неограниченной печати денег очень сложно войти в дефляционную спираль, если только не произойдет полный крах спроса и отсутствие желания печатать деньги. Дефляция в Биткойне вызвана не падением спроса, а предсказуемым ограничением предложения.

Положительным аспектом дефляции, конечно же, является ее противоположность инфляции. Инфляция вызывает медленное, но неизбежное обесценивание валюты, что приводит к скрытому налогообложению в форме наказания бережливых людей для спасения должников (включая самых крупных должников – собственно государственные органы). Контролируемые государством валюты страдают от морального риска, связанного с легкой эмиссией долговых обязательств, которые впоследствии могут быть списаны путем девальвации за счет держателей сбережений. Предстоит выяснить, является ли дефляционный аспект валюты проблемой, если он не обусловлен быстрым экономическим спадом, или преимуществом, так как защита от инфляции и девальвации перевешивает риски дефляции.

Децентрализованный консенсус

В предыдущей главе был рассмотрен блокчейн – глобальный список всех транзакций, который все участники сети Биткойн признают в качестве авторитетной записи о передаче прав собственности.

Но как все пользователи сети могут договориться о единой универсальной «правде» о том, кто чем владеет, не доверяя кому-либо? Все традиционные платежные системы зависят от модели доверия, в которой центральный орган власти предоставляет услуги клирингового центра, по сути верифицируя и проверяя все транзакции. В Биткойне нет центрального органа, однако каждый узел так или иначе хранит полную копию открытого блокчейна, которому он может доверять как авторитетной записи. Блокчейн не создается центральным органом, а собирается независимо каждым узлом в сети. Действуя на основании информации, передаваемой через незащищенные сетевые соединения, каждый узел сети таким образом приходит к одному и тому же заключению и собирает копию того же блокчейна, что и все остальные. В этой главе рассматривается процесс, с помощью которого сеть Биткойн достигает глобального консенсуса без участия центральных органов власти.

Одним из изобретений Сатоши Накамото стал децентрализованный механизм достижения *возникающего консенсуса* (*emergent consensus*). Возникающим (эмерджентным) его называют потому, что консенсус не достигается явным образом – нет никаких выборов или фиксированного момента наступления единодушного согласия (консенсуса). Вместо этого консенсус возникает как следствие асинхронного взаимодействия тысяч независимых узлов, которые следуют простым правилам. Все свойства Биткойна, включая деньги, транзакции, платежи и модель безопасности, которая не зависит от централизованной власти или доверия, проистекают из этого изобретения.

Децентрализованный консенсус Биткойна возникает в результате взаимодействия четырех независимых процессов, происходящих на узлах сети:

- независимая проверка каждой транзакции каждым полным узлом на основе всестороннего списка критериев;
- независимое объединение этих транзакций в новые блоки майнинговыми узлами в сочетании с демонстрацией вычислений алгоритмом доказательства работы (proof-of-work);

- независимая верификация новых блоков каждым узлом и их сборка в цепочку;
- независимый выбор каждым узлом цепочки с наибольшим объемом вычислений, подтвержденных алгоритмом доказательства работы.

В следующих разделах будет рассмотрено взаимодействие этих процессов для создания эмерджентного свойства консенсуса в масштабах сети. Свойства, которое позволяет любому узлу Биткойна создавать собственную копию авторитетного, надежного, открытого, глобального блокчейна.

Независимая верификация транзакций

В главе 6 был показан процесс создания транзакций программным обеспечением кошелька посредством сбора UTXO, предоставления соответствующих данных аутентификации, а затем создания новых выходов, назначенных новому владельцу. Далее полученная транзакция отправляется на соседние узлы сети для распространения по всей сети Биткойн.

Однако перед отправкой транзакций своим соседям каждый узел Биткойна, получивший транзакцию, сначала проводит ее верификацию. Это позволяет гарантировать распространение по сети только действительных транзакций, а недействительные отсеиваются на первом же встретившемся узле.

Каждый узел проверяет каждую транзакцию по обширному списку критериев:

- синтаксис и структура данных транзакции должны быть корректными;
- списки входов и выходов не должны быть пустыми;
- вес транзакции должен быть достаточно небольшим, чтобы она могла поместиться в блок;
- каждое значение выхода, как и суммарное, должно находиться в допустимом диапазоне значений (ноль или больше, но не более 21 млн биткойнов);
- время блокировки равно `INT_MAX`, или время блокировки и значения последовательности удовлетворяют правилам блокировки по времени и BIP68;
- количество операций подписи (SIGOPS), содержащихся в транзакции, меньше лимита операций подписи;
- расходуемые выходы соответствуют выходам в мемпуле или неизрасходованным выходам в блоке главной ветви;
- для каждого входа, если ссылающаяся на него выходная транзакция является выходом coinbase, она должна иметь не менее `COINBASE_MATURITY` (100) подтверждений. Также должно быть соблюдено любое абсолютное или относительное время блокировки. Узлы могут передавать транзакции за блок до наступления их срока исполнения, поскольку они будут исполнены, если будут включены в следующий блок;
- транзакция отклоняется, если сумма входных значений меньше суммы выходных значений;

- скрипты для каждого входа должны пройти валидацию по соответствующим скриптам выхода.

Обратите внимание, что со временем условия меняются, чтобы добавлять новые функции или противостоять новым атакам типа «отказ в обслуживании».

Проводя независимую процедуру верификации каждой транзакции по мере ее получения и перед дальнейшим распространением, каждый узел формирует пул действительных (но неподтвержденных) транзакций, называемый *пулом памяти* (*memory pool*), или *мемпулом* (*mempool*).

Узлы майнинга

Некоторые узлы в сети Биткойн, которые специализируются на добыче биткойнов, называют *майнерами* (*miners*). Так называемый Jing (Цзин) занимается майнингом биткойнов. Он зарабатывает биткойны с помощью «*установки для майнинга*» (*mining rig*), которая представляет собой аппаратное оборудование, специально предназначенное для этих целей. Это спецоборудование Jing для майнинга подключается к серверу, на котором работает полный узел. Как и любой другой полный узел, узел Jing получает и рассылает неподтвержденные транзакции в сети Биткойн. Однако узел Jing также объединяет эти транзакции в новые блоки.

Рассмотрим блоки, которые были созданы во время покупки Алисой товара у Боба (см. раздел «Покупка в интернет-магазине»). Для демонстрации концепций этой главы предположим, что блок с транзакцией Алисы был добыт системой майнинга Jing, и проследим за тем, как эта транзакция становится частью нового блока.

Узел майнинга Jing содержит локальную копию блокчейна. К моменту покупки Алисой чего-либо узел Jing уже достиг цепочки блоков с наибольшим количеством доказательств работы. Узел Jing отслеживает транзакции, пытается добыть новый блок, а также проверяет блоки, обнаруженные другими узлами. В процессе добычи узел Jing получает новый блок через сеть Биткойн. Появление этого блока означает окончание поиска этого блока и начало поиска следующего блока.

В течение нескольких минут, пока узел Jing искал вариант решения для предыдущего блока, он также собирал транзакции для подготовки следующего блока. К этому моменту в его пуле памяти накопилось несколько тысяч транзакций. Получив новый блок и подтвердив его, узел Jing также сравнит его со всеми транзакциями в пуле памяти и удалит из него все включенные в этот блок. Все оставшиеся в пуле памяти транзакции являются неподтвержденными и ожидают записи в новый блок.

Узел Jing немедленно формирует новый частичный блок – претендента на роль следующего блока. Этот блок называют *блоком-кандидатом* (*candidate block*), потому что он еще не является действительным блоком из-за отсутствия в нем корректного доказательства работы. Блок становится действительным только в том случае, если майнеру удается найти решение в соответствии с алгоритмом доказательства работы.

Когда узел Jing объединяет все транзакции из пула памяти, в новом блоке-кандидате оказывается несколько тысяч транзакций. Каждая из них оплачивает комиссию за транзакции, которую и попытается получить майнер.

Транзакция Coinbase

Первой транзакцией в любом блоке является специальная транзакция, которая называется *транзакцией coinbase* (*coinbase transaction*). Эта транзакция создается узлом Jing и служит для оплаты *вознаграждения* (*reward*) за майнинг.

Узел Jing создает транзакцию coinbase в качестве платежа на свой собственный кошелек. Общая сумма вознаграждения, которое Jing получает за добычу блока, складывается из субсидии на блок (6,25 нового биткойна в 2023 году) и комиссии за все включенные в блок транзакции.

В отличие от обычных транзакций, транзакция coinbase не потребляет (тратит) UTXO в качестве входных данных. Вместо этого у нее есть только один вход, называемый *входом coinbase* (*coinbase input*), в котором подразумевается вознаграждение за блок. Транзакция coinbase должна иметь как минимум один выход. Она может иметь столько выходов, сколько поместится в блок. Как правило, транзакции coinbase в 2023 году имели два выхода. Один из них был нулевым, который использовал `OP_RETURN` для фиксации всех свидетелей для транзакций с сегрегированными свидетелями (segwit) в блоке. Другой выход выплачивает вознаграждение майнеру.

Вознаграждение и комиссии coinbase

Для создания транзакции coinbase узел Jing первым делом вычисляет общую сумму комиссии за транзакцию:

Сумма комиссий = *Сумма*(*входов*) – *Сумма*(*выходов*).

Затем узел Jing вычисляет правильную сумму вознаграждения за новый блок. Вознаграждение рассчитывается в зависимости от высоты блока, начиная с 50 биткойнов за блок и уменьшаясь вдвое каждые 210 000 блоков.

Расчет можно увидеть в функции `GetBlockSubsidy` клиента Bitкойн Core, main.cpp, как показано в примере 12.3.

Пример 12.3 ❖ Расчет вознаграждения за блок

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Force block reward to zero when right shift is undefined.
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // Subsidy is cut in half every 210,000 blocks.
    nSubsidy >>= halvings;
    return nSubsidy;
}
```

Начальная субсидия рассчитывается в сатоши путем умножения 50 на константу `COIN` (100 000 000 сатоши). Таким образом, начальное вознаграждение (nSubsidy) устанавливается на уровне 5 млрд сатоши.

Далее функция вычисляет количество произошедших халвингов (`halvings`) путем деления текущей высоты блока на интервал халвинга (`SubsidyHalvingInterval`).

Потом функция использует двоичный оператор сдвига вправо для деления вознаграждения (`nSubsidy`) на два для каждого раунда халвинга. В случае с блоком 277 316 это означает, что вознаграждение в размере 5 млрд сатоши делится на два (один халвинг), и в результате получается 2,5 млрд сатоши, или 25 биткойнов. После 33-го халвинга размер вознаграждения будет округлен до нуля. Двоичный оператор сдвига вправо используется из-за его большей эффективности по сравнению с многократными повторными делениями. Чтобы избежать возможной ошибки, операция сдвига пропускается после 63 делений, а субсидия устанавливается равной 0.

Наконец, вознаграждение coinbase (`nSubsidy`) добавляется к комиссии за транзакцию (`nFees`), и сумма возвращается.

Если майнинговый узел Jing записывает транзакцию coinbase, что мешает ему «наградить» себя сотней или тысячей биткойнов? Ответ заключается в том, что завышенное вознаграждение приведет к признанию блока недействительным всеми остальными, что станет причиной пустой траты электроэнергии узлом Jing, используемой для вычисления алгоритма доказательства работы. Узел Jing может потратить вознаграждение только в том случае, если блок будет принят всеми.

Структура транзакции coinbase

После проведения этих расчетов узел Jing создает транзакцию coinbase для выплаты себе вознаграждения за блок.

Транзакция coinbase отличается особым форматом. Вместо входа транзакции, указывающего предыдущий UTXO для расходования, в ней есть вход «coinbase». Входы транзакций были рассмотрены в разделе «Входы». Теперь проведем сравнение обычного входа транзакции и входа транзакции coinbase. В табл. 12.1 показана структура обычной транзакции, а в табл. 12.2 – структура входа транзакции coinbase.

Таблица 12.1. Структура входа «обычной» транзакции

Размер	Поле	Описание
32 байта	Хеш транзакции	Указатель на транзакцию с расходуемым UTXO
4 байта	Индекс выхода	Номер индекса, расходуемого UTO, первый – 0
1–9 байт (compactSize)	Размер скрипта	Длина скрипта в байтах, см. далее
Переменный	Входной скрипт	Скрипт, выполняющий условия скрипта выхода UTXO
4 байта	Номер последовательности	Многоцелевое поле для блокировки по времени BIP68 и сигнала о замене транзакции

Таблица 12.2. Структура входа транзакции coinbase

Размер	Поле	Описание
32 байта	Хеш транзакции	Все биты равны нулю: не является хеш-ссылкой транзакции
4 байта	Индекс выхода	Все биты равны единице: 0xFFFFFFFF
1 байт	Размер данных coinbase	Длина данных coinbase, от 2 до 100 байт
Переменный	Данные coinbase	Произвольные данные для дополнительного нонса и тегов майнинга; в блоках v2 должны начинаться с высоты блока
4 байта	Номер последовательности	Устанавливается на 0xFFFFFFFF

В транзакции coinbase первые два поля имеют значения, которые не являются ссылкой на UTXO. Вместо «хеша транзакции» первое поле заполнено 32 байтами с нулевым значением. «Выходной индекс» заполняется 4 байтами, каждый из которых имеет значение 0xFF (255 в десятичном формате). Скрипт входа заменен на поле для данных coinbase. Далее будет показано, как именно это поле используется майнерами.

Данные coinbase

Транзакции coinbase не имеют поля скрипта входа. Это поле заменяется данными coinbase, размер которых может составлять от 2 до 100 байт. За исключением первых нескольких байтов, остальные данные coinbase могут использоваться майнерами по своему усмотрению; это произвольные данные.

Например, в блоке генезиса Сатоши Накамото добавил в данные coinbase текст «The Times 03/Jan/2009 Chancellor on brink of second bailout for banks», используя его в качестве доказательства самой ранней даты создания этого блока и для передачи сообщения. В настоящее время майнеры часто используют данные coinbase для включения дополнительных значений нонса и строк идентификации майнингового пула.

Первые несколько байтов coinbase раньше могли использоваться произвольно, но теперь этого нет. Согласно BIP34, блоки версии 2 (блоки с полем версии 2 или выше) должны содержать высоту блока в виде скрипта «push» в начале поля coinbase.

Построение заголовка блока

Для создания заголовка блока узел майнинга должен заполнить шесть полей, как указано в табл. 12.3.

Изначально поле версии было целочисленным и использовалось в трех обновлениях сети Биткойн, определенных в BIP 34, BIP 66 и BIP 65. Каждый раз номер версии увеличивался. В более поздних обновлениях поле версии было определено как битовое поле под названием *versionbits*, что позволило проводить до 29 обновлений одновременно; подробности см. в разделе «BIP9: сигналы и активация». Еще позже майнеры начали использовать некоторые из битов версий в качестве вспомогательного поля нонса.

Обновления протокола, определенные в BIP 34, BIP 66 и BIP 65, происходили именно в таком порядке, при этом BIP 66 (строгий DER) предшествовал BIP 65 (`OP_CHECKTIMELOCKVERIFY`), поэтому разработчики Биткойна часто перечисляют их в таком порядке, а не в очередном.

Таблица 12.3. Структура заголовка блока

Размер	Поле	Описание
4 байта	Версия	Многоцелевое битовое поле
32 байта	Хеш предыдущего блока	Ссылка на хеш предыдущего (родительского) блока в цепочке
32 байта	Корень дерева Меркла	Хеш, являющийся корнем дерева Меркла транзакций данного блока
4 байта	Временная метка	Приблизительное время создания блока (в секундах по эпохе Unix)
4 байта	Цель	Цель алгоритма доказательства работы для этого блока
4 байта	Нонс	Счетчик для алгоритма доказательства работы

Сегодня поле versionbits не имеет никакого значения, если только не проводится попытка обновления протокола консенсуса, и в этом случае необходимо прочитать его документацию для определения порядка использования versionbits.

Далее майнинговому узлу необходимо добавить «хеш предыдущего блока» (Previous Block Hash), также известный как `prevhash`. Это хеш полученного из сети заголовка блока из предыдущего блока, который узел Jing принял и выбрал в качестве *родительского* для своего блока-кандидата.

Выбирая конкретный родительский блок, на который указывает поле Previous Block Hash в заголовке блока-кандидата, узел Jing направляет свои майнинговые мощности на продление цепочки, которая заканчивается этим конкретным блоком.

Следующий шаг – закрепление всех транзакций с помощью деревьев Меркла. Каждая транзакция перечисляется по идентификатору ее свидетеля (*wtxid*) в топографическом порядке, при этом 32 байта 0x00 отводятся под wtxid первой транзакции (coinbase). Как было показано в разделе «Деревья Меркла», последний wtxid хешируется сам с собой в случае нечетного количества wtxid, в результате чего создаются узлы, каждый из которых содержит хеш одной транзакции. Затем хеши транзакций объединяются попарно, создавая каждый уровень дерева, пока все транзакции не будут сведены в один узел в «корне» (root) дерева. Корень дерева Меркла суммирует все транзакции в одно 32-байтовое значение, которое является *хешем корня свидетеля* (*witness root hash*).

Хеш корня свидетеля добавляется к выходу транзакции coinbase. Этот шаг может быть пропущен, если ни одна из транзакций в блоке не требует наличия структуры свидетеля. Затем каждая транзакция (включая транзакцию coinbase) регистрируется в списке с помощью своего идентификатора транзакции (txid) и затем используется для построения второго дерева Меркла, корень которого становится корнем Меркла с привязкой к заголовку блока.

Далее узел майнинга Jing добавляет 4-байтовую временную метку, закодированную в виде временной метки «эпохи» Unix. Она рассчитывается на осно-

ве количества секунд, прошедших с 1 января 1970 года в полночь по Гринвичу (UTC/GMT).

Потом узел Jing заполняет целевой параметр nBits, который должен быть задан как компактное представление требуемого доказательства работы для признания этого блока действительным. Значение целевого параметра хранится в блоке в виде метрики «целевые биты» (target bits), которая представляет собой кодировку целевого параметра в виде мантиссы и экспоненты. Кодировка состоит из 1-байтовой экспоненты, за которой следует 3-байтовая мантисса (коэффициент). Например, в блоке 277 316 значение целевых битов равно `0x1903a30c`. Первая часть `0x19` – это шестнадцатеричная экспонента, а следующая часть `0x03a30c` – это коэффициент. Концепция цели объясняется в разделе «Ретаргетинг для корректировки сложности», а представление «целевых битов» – в разделе «Представление цели».

Последнее поле – это нонс, который имеет нулевую инициализацию.

После заполнения всех остальных полей заголовок блока-кандидата полностью сформирован, и можно приступать к процессу майнинга. Теперь задача состоит в поиске заголовка, при котором хеш будет меньше целевого. Узлу майнинга придется проверить миллиарды или триллионы вариантов заголовка, прежде чем будет найдена удовлетворяющая требованиям версия.

Майнинг блока

Теперь, когда узел Jing создал блок-кандидат, пришло время работы аппаратного майнингового оборудования Jing для «добычи» блока посредством решения алгоритма доказательства работы. Именно это решение сделает блок действительным. На протяжении всей этой книги рассматриваются криптографические хеш-функции, используемые в различных аспектах системы Биткойн. В процессе майнинга Биткойна используется хеш-функция SHA256.

Проще говоря, майнинг – это процесс многократного хеширования заголовка блока-кандидата с изменением одного параметра, до тех пор, пока полученный хеш не совпадет с определенной целью. Результат хеш-функции нельзя определить заранее, как нельзя создать шаблон для получения конкретного значения хеша. Эта особенность хеш-функций означает, что единственный способ получить результат хеша, соответствующий определенной цели, – это повторение попыток снова и снова, каждый раз изменяя входные данные до тех пор, пока нужный результат хеша не будет получен случайным образом.

Алгоритм доказательства работы

На вход хеш-алгоритма подаются данные произвольной длины, а на выходе получается детерминированный результат фиксированной длины, называемый *дайджестом* (*digest*). Этот дайджест представляет собой цифровое обязательство по отношению к входным данным. Для любого конкретного входа результирующий дайджест всегда будет одинаковым и может быть легко вычислен и проверен каждым, кто применяет тот же хеш-алгоритм. Ключевой особенно-

стью криптографического хеш-алгоритма является невозможность вычислить два разных входа, дающих одинаковый дайджест (такая ситуация называется *коллизией* – *collision*). Как следствие также практически невозможно подобрать входные данные таким образом, чтобы получить желаемый дайджест. Это можно сделать только путем перебора случайных входных данных.

При использовании SHA256 выход всегда имеет длину 256 бит, независимо от размера входа. Для примера вычислим хеш SHA256 фразы «Hello, World!»:

```
$ echo "Hello, world!" | sha256sum
d9014c4624844aa5bac314773d6b689ad467fa4e1d1a50a1b8a99d5a95f72ff5 -
```

Этот 256-битный выход (в шестнадцатеричном виде) является хешем или дайджестом фразы и зависит от каждой ее части. Добавление одной буквы, знака препинания или любого другого символа приведет к получению иного хеша.

Переменная, используемая в таком сценарии, называется нонс (nonce). Нонс используется для изменения выхода криптографической функции, а в данном случае – для изменения выхода SHA256-обязательства к этой фразе.

Чтобы усложнить алгоритм, определим цель: найти фразу, которая дает шестнадцатеричный хеш, начинающийся с нуля. К счастью, это несложно, как показано в примере 12.4.

Пример 12.4 ❖ Простая реализация доказательства работы

```
$ for nonce in $( seq 100 ) ; do echo "Hello, world! $nonce" | sha256sum ; done
3194835d60e85bf7f728f3e3f4e4e1f5c752398cbcc5c45e048e4dbcae6be782 -
bfa474bbe2d9626f578d7d8c3acc1b604ec4a7052b188453565a3c77df41b79e -
[...]
f75a100821c34c84395403afd1a8135f685ca69ccf4168e61a90e50f47552f61 -
09cb91f8250df04a3db8bd98f47c7cecb712c99835f4123e8ea51460ccbec314 -
```

Фраза «Hello, World! 32» дает следующий хеш, который подходит под заданные нами критерии: `09cb91f8250df04a3db8bd98f47c7cecb712c99835f4123e8ea51460ccbec314`. На его поиски понадобилось 32 попытки. С точки зрения вероятности, если выход хеш-функции распределен равномерно, то можно ожидать, что результат с 0 в качестве шестнадцатеричного префикса будет найден раз в 16 хешей (одна из 16 шестнадцатеричных цифр от 0 до F). В числовом выражении это означает поиск хеш-значения менее `0x1000000000000000000000000000000000000000000000000000000000000000`. Этот пороговый показатель называют *целью* (*target*), и задача состоит в том, чтобы найти хеш, который численно меньше этой цели. Если уменьшить цель, задача поиска хеша, который меньше этой цели, становится все сложнее и сложнее.

Чтобы привести простую аналогию, представьте себе игру, в которой игроки бросают пару костей несколько раз, стараясь выбросить меньше заданной цели. В первом раунде цель – 12. Если вы не выбросите двойную шестерку, вы выиграете. В следующем раунде цель – 11. Чтобы выиграть, игроки должны выбросить 10 или меньше, что, опять же, является простой задачей. Допустим, через несколько раундов цель снизится до 5. Теперь более половины выпавших очков превысят цель и, следовательно, будут считаться неудачными. Чем меньше цель, тем больше бросков кубиков требуется для победы. В конце концов,

когда цель равна 3 (минимально возможное значение), только один бросок из каждых 36, или около 3 %, даст выигрышный результат.

С точки зрения наблюдателя, который осведомлен, что целью игры является 3, при успешном выпадении выигрышного числа очков можно предположить, что он сделал в среднем 36 бросков. Другими словами, по сложности цели можно оценить объем работы, необходимый для достижения успеха. Если алгоритм основан на детерминированной функции, такой как SHA256, то сам входной сигнал является *доказательством* (*proof*) того, что для получения результата ниже целевого было проделано определенное количество работы. Поэтому его и называют *доказательством работы* (*proof of work*).

Несмотря на случайный исход каждой попытки, вероятность любого возможного результата можно рассчитать заранее. Поэтому результат определенной сложности является доказательством выполнения определенного объема работы.

В примере 12.4 выигрыш «нонса» равен 32, и этот результат может быть независимо подтвержден любым желающим. Любой может добавить число 32 в качестве суффикса к фразе «Hello, world!» и вычислить хеш, удостоверившись, что он меньше заданной цели:

```
$ echo "Hello, world! 32" | sha256sum
09cb91f8250df04a3db8bd98f47c7cecb712c99835f4123e8ea51460ccbec314
```

Хотя для проверки требуется всего одно вычисление хеша, нам потребовалось 32 вычисления хеша для поиска подходящего нонса. Если бы цель была ниже (сложность выше), для подбора подходящего нонса потребовалось бы гораздо больше хеш-вычислений, но для проверки достаточно всего одного хеш-вычисления. Зная цель, любой может подсчитать сложность с помощью статистики и, следовательно, приблизительно определить объем работы, необходимый для поиска такого нонса.

Доказательство работы должен выдать хеш, значение которого меньше целевого. Более высокая цель означает, что найти хеш меньше целевого будет менее сложно. Более низкая цель означает, что найти хеш ниже целевого значения сложнее. Цель и сложность находятся в обратной зависимости.

Доказательство работы в Биткойне очень похоже на задачу из примера 12.4. Майнер создает блок-кандидат, заполненный транзакциями. Затем майнер вычисляет хеш заголовка этого блока и смотрит, меньше ли он, чем *нынешняя цель*. Если хеш не меньше цели, майнер изменяет нонс (обычно просто увеличивая его на единицу) и повторяет попытку. При текущей сложности сети Биткойн майнерам биткойнов приходится повторять огромное количество попыток, прежде чем удается найти нонс, дающий достаточно низкий хеш заголовка блока.

Отображение цели

Заголовки блоков содержат цель в нотации, которую называют «*целевые биты*» (*target bits*), или просто «*биты*» (*bits*). В блоке 277 316 эта цель имеет значение `0x1903a30c`. Эта нотация выражает цель доказательства работы в формате ко-

эффициент/экспонента, где первые две шестнадцатеричные цифры – экспонента, а следующие шесть шестнадцатеричных цифр – коэффициент. Таким образом, в данном блоке экспонента равна `0x19`, а коэффициент – `0x03a30c`.

Формула для расчета сложности цели из этого отображения такова:

$$\text{цель} = \text{коэффициент} \times 2^{(8 \times (\text{экспонента} - 3))}.$$

Используя эту формулу и значение битов сложности 0x1903a30c, получается:

$$\text{цель} = 0\text{x}03\text{a}30\text{c} \times 2^{0\text{x}08 \times (0\text{x}19 - 0\text{x}03)},$$

что составляет:

22 829 202 948 393 929 850 749 706 076 701 368 331 072 452 018 388 575 715 328.

Или в шестнадцатеричном виде:

0x00000000000000003A30C0000000000000000000000000000000000000000000.

Это означает, что действительным блоком для высоты 277 316 является тот, у которого хеш заголовка блока меньше заданного целевого значения. В двоичном виде это число должно иметь более 60 старших битов, установленных в ноль. При таком уровне сложности один майнер, обрабатывающий 1 триллион хешей в секунду (1 терахешей в секунду, или 1 TH/sec), будет находить решение в среднем лишь раз на 8496 блоков, или раз в 59 дней.

Ретаргетинг для корректировки сложности

Как было показано выше, цель определяет сложность и, следовательно, влияет на время, необходимое для поиска решения алгоритма доказательства работы. Это приводит к очевидным вопросам: почему сложность можно регулировать, кто и как ее регулирует?

Блоки Биткойна генерируются в среднем каждые 10 минут. Это «сердцебиение» Биткойна, которое определяет частоту эмиссии валюты и скорость расчетов по транзакциям. Она должна оставаться постоянной не только в краткосрочной перспективе, но и на протяжении многих десятилетий. Ожидается, что на протяжении этого времени мощность компьютеров будет стремительно увеличиваться. Кроме того, количество участников процесса майнинга и используемые ими компьютеры также будут постоянно изменяться. Чтобы время генерации блока не превышало 10 минут, сложность майнинга должна быть скорректирована с учетом этих изменений. По сути, цель доказательства работы – это динамический параметр, который периодически корректируется для соблюдения цели 10-минутного интервала между блоками. Проще говоря, цель устанавливается таким образом, чтобы при нынешней мощности майнинга интервал между блоками составлял 10 минут.

Как же осуществляется такая корректировка в полностью децентрализованной сети? Ретаргетинг («перенацеливание» – retargeting) происходит автоматически и на каждом узле независимо. Через каждые 2016 блоков все узлы производят ретаргетинг доказательства работы. Вычисляется соотношение между фактическим и желаемым промежутками времени в 10 минут на блок, и производится пропорциональная корректировка целевого показателя (в сторону

увеличения или уменьшения). Проще говоря, если сеть находит блоки быстрее, чем раз в 10 минут, сложность увеличивается (цель уменьшается). Если процесс поиска блоков происходит медленнее, чем ожидалось, сложность уменьшается (цель увеличивается).

Уравнение можно обобщить следующим образом:

```
Новая цель = Старая цель * (20 160 минут / Фактическое время последних 2015 блоков).
```

Хотя целевая калибровка происходит каждые 2016 блоков из-за ошибки смещения на единицу в исходном программном обеспечении Биткойна, она опирается на общее время предыдущих 2015 блоков (а не 2016, как должно быть), что приводит к перекосу в сторону более высокой сложности на 0,05 %.

В примере 12.5 показан код, используемый в клиенте Bitcoin Core (`CalculateNextWorkRequired()` в pow.cpp).

Пример 12.5 ❖ Ретаргетинг доказательства работы

```
// Limit adjustment step
 int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
 LogPrintf(" nActualTimespan = %d before bounds\n", nActualTimespan);
 if (nActualTimespan < params.nPowTargetTimespan/4)
     nActualTimespan = params.nPowTargetTimespan/4;
 if (nActualTimespan > params.nPowTargetTimespan*4)
     nActualTimespan = params.nPowTargetTimespan*4;

 // Retarget
 const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
 arith_uint256 bnNew;
 arith_uint256 bnOld;
 bnNew.SetCompact(pindexLast->nBits);
 bnOld = bnNew;
 bnNew *= nActualTimespan;
 bnNew /= params.nPowTargetTimespan;

 if (bnNew > bnPowLimit)
     bnNew = bnPowLimit;
```

Параметры `Interval` (2016 блоков) и `TargetTimespan` (две недели, или 1 209 600 секунд) задаются в *chainparams.cpp*.

Чтобы избежать экстремальных колебаний сложности, корректировка ретаргетинга должна производиться менее чем четырежды (4) за цикл. Если требуемая корректировка цели превышает четыре раза, она будет скорректирована в 4 раза и не более. Любая дальнейшая корректировка будет выполнена в следующем периоде ретаргетинга, поскольку дисбаланс сохранится в течение следующих 2016 блоков. Поэтому большие расхождения между мощностью хеширования и сложностью могут потребовать нескольких циклов по 2016 блоков для выравнивания.

Обратите внимание, что цель не зависит ни от количества транзакций, ни от их стоимости. Это означает, что количество мощностей хеширования и, соответственно, электроэнергии, затрачиваемой на обеспечение безопасности Биткойна, также не зависит от количества транзакций. Биткойн может масштабироваться и оставаться безопасным без какого-либо увеличения мощно-

сти хеширования относительно нынешнего уровня. Увеличение мощности хеширования является результатом действия рыночных сил, поскольку на рынок выходят новые майнеры. Пока достаточное количество хеш-мощностей находится под контролем майнеров, честно работающих в стремлении получить вознаграждение, этого достаточно для предотвращения атак «захвата» (takeover) и, следовательно, для обеспечения безопасности Биткойна.

Сложность майнинга тесно связана со стоимостью электроэнергии и курсом биткойна по отношению к валюте, используемой для оплаты электроэнергии. Высокопроизводительные системы для майнинга настолько эффективны, насколько это возможно при нынешнем поколении кремниевых технологий. Они преобразуют электроэнергию в вычисления хеширования с максимально возможной скоростью. Основное влияние на рынок майнинга оказывает цена одного киловатт-часа электроэнергии в биткойнах, поскольку она определяет рентабельность майнинга и, следовательно, стимулы для вхождения или выхода с рынка майнинга.

Медианное время прошедшего периода (MTP)

В Биткойне существует тонкая, но очень существенная разница между временем на часах и временем консенсуса. Биткойн – это децентрализованная сеть, а значит, каждый участник имеет свое собственное представление о времени. События в сети не происходят везде моментально. Задержки в сети необходимо учитывать в концепции каждого узла. В конечном итоге все синхронизируется для создания общего блокчейна. Каждые 10 минут Биткойн достигает консенсуса относительно состояния блокчейна в том виде, в котором он существовал в *прошлом*.

Временные метки в заголовках блоков устанавливаются майнерами. Правила консенсуса допускают определенную свободу действий, чтобы учесть разницу в точности часов между децентрализованными узлами. Однако для майнеров это создает нежелательный стимул лгать о времени в блоке. Например, если майнер устанавливает время в будущем, он может снизить сложность, что позволит ему добыть больше блоков и получить часть субсидии на блок, зарезервированной для будущих майнеров. Если они могут установить для некоторых блоков время в прошлом, они могут использовать текущее время для других блоков, и, таким образом, опять создать видимость большого промежутка времени между блоками для манипуляций со сложностью.

Для предотвращения манипуляций в Биткойне действуют два правила консенсуса. Первое заключается в приеме блока со временем, отдаленным в будущее более чем на два часа. Второе – ни один узел не примет блок со временем, меньшим или равным медианному времени последних 11 блоков. Его называют *медианным временем прошедшего периода* (*Median Time Past*, MTP).

Вместе с введением относительных блокировок по времени BIP68 также изменился способ расчета «времени» для этих блокировок (как абсолютных, так и относительных) в транзакциях. Это стимулировало майнеров использовать самое позднее из возможного времени (близкое к двум часам в будущем), чтобы в блок попало больше транзакций.

Чтобы избавиться от стимула лгать и для усиления безопасности блокировок по времени, был предложен BIP113, который был активирован одновременно с другими протоколами BIP для относительных блокировок по времени. Значение MTP стало временем консенсуса, используемым для всех расчетов блокировок времени. Взяв средний момент примерно за два часа до этого, можно уменьшить влияние временной метки любого отдельного блока. Благодаря использованию в расчетах 11 блоков ни один майнер не может влиять на временные метки для получения прибыли от транзакций с временными метками, которые еще не достигли «зрелости».

MTP меняет процесс вычисления времени блокировки (calculations for lock time), CLTV (Check Lock Time Verify), последовательности и CSV (Relative Timelocks). Время консенсуса, рассчитываемое с помощью MTP, обычно отстает от реального времени примерно на один час. Если вы создаете транзакции с блокировкой по времени, вам следует учитывать это при оценке желаемого значения для кодирования времени блокировки, последовательности, CLTV и CSV.

Успешный майнинг блока

Как было сказано ранее, узел Jing создает блок-кандидат и готовит его к майнингу. Узел Jing оснащен несколькими аппаратными установками майнинга со специализированными микросхемами, на которых сотни тысяч процессоров параллельно обрабатывают двойной алгоритм Биткойна SHA256 с невероятной скоростью. Многие из этих специализированных систем подключены к его узлу майнинга через USB или локальную сеть. Затем узел майнинга, запущенный на рабочем столе узла Jing, передает заголовок блока на его оборудование для майнинга. Там начинается тестирование триллионов вариантов заголовка в секунду. Так как нонс состоит всего из 32 бит, после перебора всех возможных вариантов нонса (около 4 миллиардов) оборудование для майнинга изменяет заголовок блока (корректируя дополнительное место для нонса, биты версии или временную метку coinbase) и сбрасывает счетчик нонса, тестируя новые комбинации.

Спустя почти 11 минут после начала добычи конкретного блока одна из аппаратных машин майнинга находит решение и отправляет его обратно на узел майнинга.

Тут же узел майнинга Jing передает блок всем своим пирам. Они получают его, проверяют, а затем распространяют этот новый блок. По мере распространения блока по сети каждый узел добавляет его в свою копию блокчейна, поднимая его до нового уровня высоты. Когда узлы майнинга получают и подтверждают блок, они прекращают собственные попытки поиска блока на той же высоте и немедленно начинают вычислять следующий блок в цепочке с использованием блока Jing в качестве «родительского». Выстраивая работу поверх только что обнаруженного блока Jing, другие майнеры, по сути, используют свои мощности майнинга для подтверждения блока Jing и расширяемой им цепочки.

В следующем разделе будет рассмотрен процесс, с помощью которого каждый узел подтверждает блок и выбирает наиболее работоспособную цепочку, создавая консенсус, формирующий децентрализованный блокчейн.

Проверка нового блока

Третий шаг в механизме консенсуса Биткойна – независимая верификация каждого нового блока всеми узлами сети. По мере продвижения нового блока по сети каждый узел выполняет серию тестов для его валидации. Независимая проверка также гарантирует включение в блокчейн только тех блоков, которые соответствуют правилам консенсуса, что приносит вознаграждение их майнерам. Блоки, нарушающие правила, отклоняются и не только лишают майнеров вознаграждения, но также приводят к напрасным затратам усилий на поиск решения для доказательства работы, в результате чего майнеры несут все расходы по созданию блока, но не получают никакой выгоды.

После получения нового блока узел проверяет его на соответствие длинному списку обязательных критериев. В противном случае блок отклоняется. Эти критерии можно увидеть в функциях `CheckBlock` и `CheckBlockHeader` клиента Bitcoin Core:

- структура данных блока синтаксически корректна;
- хеш заголовка блока меньше целевого (обеспечивает доказательство работы);
- временная метка блока находится между MTP и двумя часами в будущем (с учетом временных погрешностей);
- вес блока находится в допустимых пределах;
- первая транзакция (и только первая) – это транзакция coinbase;
- все транзакции в блоке являются действительными в соответствии с контрольным списком транзакций, рассмотренным в разделе «Независимая верификация транзакций».

Независимое подтверждение каждого нового блока каждым узлом сети гарантирует, что майнерам не удастся сжульничать. В предыдущих разделах был описан процесс написания майнерами транзакции, которая приносит им вознаграждение в виде новых биткойнов, созданных в блоке, а также комиссию за транзакцию. Почему бы майнерам не написать себе транзакцию на тысячу биткойнов вместо законного вознаграждения? Потому что каждый узел проверяет блоки по одним и тем же правилам. Недействительная транзакция coinbase сделает недействительным весь блок, в результате чего он будет отклонен, и, следовательно, эта транзакция никогда не станет частью блокчейна. Майнерам необходимо создать блок на основе общих правил, которым следуют все узлы, и добыть его с правильным решением доказательства работы. Для этого им приходится расходовать много электроэнергии, и в случае обмана вся эта электроэнергия и все усилия будут потрачены впустую. Именно поэтому независимая верификация является ключевым компонентом децентрализованного консенсуса.

Сборка и выбор цепочек блоков

Последняя часть механизма децентрализованного консенсуса в Биткойне – это объединение блоков в цепочки и выбор цепочки с наибольшим количеством доказательств работы.

Лучший блокчейн (*best blockchain*) – это та цепочка блоков, которая имеет наибольший кумулятивный уровень связанного с ней доказательства работы. Лучшая цепочка также может иметь ветви с блоками, которые являются «братьями и сестрами» (siblings) блоков, входящих в лучшую цепочку. Эти блоки действительны, но не являются частью лучшей цепочки. Они хранятся для будущего использования на тот случай, если одна из этих вторичных цепочек впоследствии станет основной. Когда появляются блоки-«братья», они обычно являются результатом почти одновременной добычи разных блоков на одной и той же высоте.

При получении нового блока узел пытается добавить его в существующий блокчейн. Узел проверит поле «хеш предыдущего блока» в этом блоке, которое является ссылкой на родительский блок. Затем узел попытается найти этого родителя в существующем блокчейне. В большинстве случаев родитель будет «вершиной» лучшей цепочки, то есть этот новый блок продолжит лучшую цепочку.

Иногда новый блок не продлевает лучшую цепочку. В этом случае узел прикрепляет заголовок нового блока к вторичной цепочке, а затем сравнивает работу этой вторичной цепочки с предыдущей лучшей цепочкой. Если вторичная цепочка теперь является лучшей, узел реорганизует свое представление о подтвержденных транзакциях и доступных UTXO. Если узел является майнером, то теперь он будет создавать блок-кандидат на продление этой новой цепочки с более высоким доказательством работы.

Выбирая цепочку с наибольшим накопленным доказательством работы, все узлы в сети в конечном итоге достигают консенсуса. Временные расхождения между цепочками в конечном итоге устраняются, так как добавляется больше работы, продлевающей одну из возможных цепочек.

Форки (развилки) блокчейна, описанные в этом разделе, происходят естественным образом в результате задержек передачи данных в глобальной сети. Далее в этой главе также будут рассмотрены преднамеренно спровоцированные форки.

Форки почти всегда разрешаются в пределах одного блока. Случайный форк может распространиться на два блока, если оба этих блока будут найдены майнерами почти одновременно с противоположных «сторон» предыдущего форка. Однако вероятность этого невелика. Интервал между блоками в Биткойне составляет 10 минут, что является компромиссом между быстрым временем подтверждения и вероятностью форка. При более быстром времени блокировки транзакции будут проходить быстрее, но это приведет к более частым форкам блокчейна, в то время как более медленное время блокировки уменьшит количество форков, но приведет к замедлению расчетов.

Что более безопасно: транзакция в одном блоке, где среднее время между блоками составляет 10 минут, или транзакция в блоке, поверх которого построено девять блоков, где среднее время между блоками составляет одну минуту? Ответ заключается в том, что

они одинаково безопасны. Злонамеренному майнеру для двойного расходования этой транзакции потребуется выполнить объем работы, эквивалентный 10 минутам от общего хешрейта сети, чтобы создать цепочку с равным доказательством работы.

Сокращение времени между блоками не приводит к более быстрым расчетам. Единственное его преимущество – предоставление более низких гарантий тем, кто готов их принять. Например, если вы готовы принять трехминутное согласие майнеров на лучший блокчейн в качестве достаточной безопасности, вы предпочтете систему с 1-минутными блоками, где вы будете ждать три блока, а не систему с 10-минутными блоками. Чем короче время между блоками, тем больше работы майнеров тратится на случайные форки (в дополнение к другим проблемам), поэтому многие предпочитают 10-минутные блоки Биткойна более коротким интервалам между блоками.

Майнинг и лотерея хешей

Майнинг биткойнов – чрезвычайно конкурентная отрасль. С каждым годом существования Биткойна мощность хеширования увеличивалась в геометрической прогрессии. В отдельные годы этот рост отражал полную смену технологий, как, например, в 2010 и 2011 годах, когда многие майнеры перешли от майнинга на центральных процессорах (CPU) к майнингу на графических ускорителях (GPU) и программируемых вентильных матрицах (FPGA). В 2013 году внедрение майнинга на специализированных чипах ASIC привело к очередному гигантскому прорыву в мощности майнинга, поскольку функция double-SHA256 была размещена непосредственно на кремниевых чипах, специализированных для майнинга. Первые такие чипы могли обеспечить больше мощности майнинга в одном корпусе, чем вся сеть Биткойна в 2010 году.

На момент написания этой книги считалось, что гигантских прорывов в развитии оборудования для майнинга биткойнов уже не будет, поскольку отрасль достигла пределов закона Мура. Согласно этому закону, производительность вычислений удваивается примерно каждые 18 месяцев. Тем не менее мощности майнинга в сети продолжают развиваться быстрыми темпами.

Решение проблемы дополнительного нонса

После 2012 года майнинг развивается в направлении устранения фундаментального ограничения в структуре заголовка блока. В первые дни существования Биткойна майнеры биткойнов могли найти блок посредством перебора значений нонса до тех пор, пока полученный хеш не окажется ниже целевого. По мере увеличения сложности майнерам зачастую приходилось перебирать все 4 млрд значений нонса, не находя блока. Однако эта проблема легко решалась путем обновления временной метки блока с учетом прошедшего времени. Поскольку временная метка является частью заголовка, это изменение позволило бы майнерам снова перебрать все значения нонса с разными результатами. Но когда скорость оборудования для майнинга превысила 4 млрд хешей в секунду (GH/sec), такой подход становился все более сложным, так как значения нонса исчерпывались менее чем за секунду. Поскольку оборудование для майнинга на базе ASIC стало превышать хешрейт в триллионы хешей в се-

кунду (TH/sec), программному обеспечению для майнинга требовалось больше места для значений нонса для поиска корректных блоков. Временную метку можно было немного растянуть, но слишком большой сдвиг в будущее приводил к признанию блока недействительным. Потребовался новый источник изменений в заголовке блока.

Одним из решений, получивших широкое распространение, стало использование транзакции coinbase в качестве источника дополнительных значений нонса. Поскольку скрипт coinbase может хранить от 2 до 100 байт данных, майнеры стали использовать это пространство в качестве дополнительного места для нонса, что позволило им анализировать гораздо больший диапазон значений заголовка блока для поиска корректных блоков. Транзакция coinbase включена в дерево Меркла, поэтому любое изменение в скрипте coinbase приводит к изменению корня Меркла. Восемь байт дополнительного нонса плюс 4 байта «стандартного» нонса позволяют майнерам исследовать в общей сложности 2^{96} (8 и далее 28 нулей) вариантов в секунду без необходимости изменения временной метки.

Другое широко распространенное на сегодняшний день решение – задействовать для майнинга до 16 бит поля versionbits заголовка блока, как описано в BIP320. Если у каждой единицы майнингового оборудования есть своя транзакция coinbase, это позволяет каждому отдельному устройству для майнинга выполнять работу с производительностью до 281 терахеша в секунду (TH/sec), внося изменения только в заголовок блока. Это упрощает оборудование и протоколы майнинга, а не увеличивает дополнительный нонс в транзакции coinbase каждые 4 млрд хешей, что требует пересчета всего левого фланга дерева Меркла до самого корня.

Пулы майнинга

В таких условиях жесткой конкуренции у майнеров, работающих в одиночку (также известных как майнеры-одиночки – solo miners), не остается ни единого шанса. Вероятность найти блок, который компенсирует их затраты на электроэнергию и оборудование, настолько мала, что представляет собой азартную игру, подобную игре в лотерею. Даже самая быстрая домашняя система ASIC для майнинга не в состоянии угнаться за коммерческими операциями, которые складывают десятки тысяч таких систем на гигантских складах рядом с электростанциями. Сейчас многие майнеры объединяются в пулы майнинга, собирая хеш-мощности и распределяя вознаграждение между тысячами участников. Участвуя в пуле, майнеры получают меньшую долю от общего вознаграждения, зато, как правило, получают вознаграждение каждый день, что уменьшает степень неопределенности.

Рассмотрим конкретный пример. Допустим, майнер приобрел оборудование для майнинга с суммарным хешрейтом 0,0001 % от текущего общего хешрейта сети. Если сложность протокола никогда не изменится, этот майнер будет находить новый блок примерно раз в 20 лет. Это очень долгий срок для ожидания оплаты. Однако если он будет работать в пуле майнеров вместе с другими майнерами, чей суммарный хешрейт составляет 1 % от общего хешрейта сети, они

будут добывать в среднем более одного блока в день. Этот майнер будет получать только свою часть вознаграждения (за вычетом любых комиссий, взимаемых пулом), поэтому в день он будет получать лишь небольшую сумму. Если бы они майнили каждый день в течение 20 лет, они заработали бы столько же (не считая комиссии пула), сколько и при самостоятельном поиске среднего блока. Единственное принципиальное различие заключается в частоте получаемых ими выплат.

Пулы майнинга координируют работу сотен или тысяч майнеров с помощью специализированных протоколов пула. Отдельные майнеры настраивают свое майнинговое оборудование на подключение к серверу пула после создания учетной записи в нем. Во время добычи их оборудование остается подключенным к серверу пула, синхронизируя свои действия с другими майнерами. Таким образом, майнеры пула разделяют между собой работу по майнингу блока и получают за это вознаграждение.

За успешные блоки вознаграждение выплачивается на биткойн-адрес пула, а не отдельным майнерам. Сервер пула периодически производит выплаты на майнерские биткойн-адреса, как только их доля вознаграждения достигает определенного порога. Как правило, сервер пула взимает процентную комиссию от вознаграждения за предоставление услуги пула майнеров.

Майнеры, участвующие в пуле, делят между собой работу по поиску решения для блока-кандидата, получая «доли» (shares) за свой вклад в процесс майнинга. Пул майнинга устанавливает более высокую цель (более низкую сложность) для получения доли, как правило, более чем в 1000 раз легче, чем цель сети Биткойн. Когда кто-то из пула успешно добывает блок, вознаграждение получает пул, а затем оно распределяется между всеми майнерами пропорционально количеству долей, которые они внесли в эту работу.

Многие пулы открыты для участия любого майнера, большого или маленького, профессионала или любителя. Поэтому в пуле могут быть как участники с одной небольшой машиной для майнинга, так и обладатели целого парка высококлассного оборудования для майнинга. Одни майнят, потребляя несколько десятков киловатт электроэнергии, другие управляют дата-центром с мегаваттным энергопотреблением. Как пулу измерить индивидуальный вклад для справедливого распределения вознаграждения без вероятности обмана? Ответ заключается в использовании алгоритма доказательства работы Биткойна для измерения вклада каждого майнера пула, но на более низкой сложности, чтобы даже самые небольшие майнеры биткойнов выигрывали долю так часто, чтобы вклад в пул был оправдан. Установив более низкую сложность для получения доли, пул измеряет объем работы, проделанной каждым майнером. Каждый раз, когда майнер пула находит хеш заголовка блока, который меньше целевого значения пула, он доказывает, что проделал работу по хешированию для нахождения этого результата. Этот заголовок в конечном итоге фиксируется в транзакции coinbase и может быть использован для доказательства использования майнером транзакции coinbase, которая выплатила бы пулу вознаграждение за блок. Каждому майнеру пула предоставляется немного отличающийся шаблон транзакции coinbase, поэтому каждый из них хеширует разные заголовки блоков-кандидатов, предотвращая дублирование усилий.

Работа по поиску долей вносит статистически измеримый вклад в общие усилия по поиску хеша ниже целевого значения сети Биткойн. Тысячи майнеров, пытающихся найти хеши с низким значением, в конце концов находят достаточно низкий хеш для удовлетворения цели сети Биткойн.

Вспомним аналогию с игровыми кубиками. Если игроки в кости бросают кубики с целью выбросить меньше четырех (общая сложность сети), то пул ставит более легкую цель, подсчитывая, сколько раз игрокам пула удается выбросить меньше восьми. Когда игроки пула выигрывают меньше восьми (цель пула), они зарабатывают доли, но не выигрывают игру, потому что не достигают цели игры (меньше четырех). Игроки в пуле гораздо чаще достигают более легкой цели, что позволяет им регулярно зарабатывать доли, даже не достигая более сложной цели – победы в игре. Время от времени один из игроков пула выбрасывает на кубиках меньше четырех, и пул выигрывает. Тогда заработок может быть распределен между игроками пула в соответствии с заработанными ими долями. Даже если цель «восемь или меньше» не была выигрышной, это был справедливый способ измерения бросков кубиков для игроков, и он иногда давал бросок меньше четырех.

Точно так же пул майнинга устанавливает (более высокую и легкую) цель пула, которая гарантирует, что отдельный майнер будет часто зарабатывать доли при нахождении хешей заголовков блоков, которые меньше цели пула. Время от времени в результате одной из таких попыток будет получен хеш заголовка блока, который меньше цели сети Биткойн, что сделает его действительным блоком, и весь пул выиграет.

Управляемые пулы

Большинство пулов майнинга являются «управляемыми» (managed). Это означает, что сервером пула управляет компания или частное лицо. Владелец сервера пула называется оператором пула (pool operator), и он взимает с майнеров пула определенный процент от дохода.

На сервере пула работает специализированное программное обеспечение и протокол майнинга пула, который координирует деятельность его майнеров. Сервер пула также подключается к одному или нескольким полным узлам Биткойна. Это позволяет серверу пула подтверждать блоки и транзакции от имени майнеров пула, освобождая их от необходимости управлять полноценным узлом. Для некоторых майнеров возможность майнить без использования полноценного узла является еще одним преимуществом участия в управляемом пуле.

Майнеры пула подключаются к серверу пула с помощью протокола майнинга, например Stratum (версии 1 или 2). Stratum v1 создает шаблоны блоков, которые содержат шаблон заголовка блока-кандидата. Сервер пула создает блок-кандидат путем агрегирования транзакций, добавления транзакции coinbase (с дополнительным пространством для нонса), вычисления корня Меркла и соединения с хешем предыдущего блока. Затем заголовок блока-кандидата отправляется каждому майнеру пула в качестве шаблона. Далее каждый майнер пула добывает блок по шаблону с более высокой (легкой) целью, чем цель сети Биткойн, и отправляет все успешные результаты обратно на сервер пула для получения своей доли дохода.

Stratum v2 опционально позволяет отдельным майнерам в пуле самим выбирать транзакции для своих блоков. Они могут отбирать их с помощью своего собственного полного узла.

Пиринговый пул майнинга (P2Pool)

Управляемые пулы с использованием Stratum v1 допускают возможность махинаций со стороны оператора пула. Он может направить усилия пула на проведение транзакций двойного расходования или аннулирование блоков (см. раздел «Атаки на хешрейт»). Кроме того, централизованные серверы пулов представляют собой единую точку отказа (single point of failure). Если сервер пула выходит из строя или замедляется в результате атаки типа «отказ в обслуживании», майнеры пула лишаются возможности осуществлять майнинг. В 2011 году для решения проблем подобной централизации был предложен и реализован новый принцип работы пула: P2Pool (peer-to-peer mining pool), одноранговый пул майнинга без центрального оператора.

Принцип работы P2Pool заключается в децентрализации функций сервера пула и реализации параллельной системы наподобие блокчейна, называемой *долевой цепочкой* (*share chain*). Цепочка долей – это блокчейн с более низкой сложностью, чем блокчейн Биткойна. Цепочка долей позволяет майнерам сотрудничать в децентрализованном пуле, добывая доли цепочки долей со скоростью один блок долей каждые 30 секунд. В каждом из блоков цепочки долей записывается пропорциональное вознаграждение майнеров пула, которые вносят свой вклад в его работу и переносят доли с предыдущего долевого блока. Когда один из долевых блоков достигает цели сети Биткойн, он передается и включается в блокчейн Биткойна, вознаграждая всех майнеров пула, которые участвовали во всех долевых блоках, предшествовавших победившему долевому блоку. По сути, вместо использования сервера пула для отслеживания долей и вознаграждений для майнеров цепочка долей позволяет всем майнерам пула отслеживать все доли с помощью децентрализованного механизма консенсуса, подобного механизму консенсуса блокчейна в Биткойне.

Майнинг P2Pool сложнее, чем майнинг пула, поскольку требует от майнеров наличия отдельного компьютера с достаточным дисковым пространством, памятью и пропускной способностью интернета для поддержки полного узла Биткойна и программного обеспечения узла P2Pool. Майнеры P2Pool подключают свое оборудование для майнинга к локальному узлу P2Pool, который выполняет функции сервера пула, отправляя шаблоны блоков на оборудование для майнинга. В сети P2Pool отдельные майнеры пула создают свои собственные блоки-кандидаты, агрегируя транзакции, как и одиночные майнеры, но затем осуществляют совместный майнинг в общей цепочке. P2Pool – это гибридный подход, преимущество которого заключается в более детализированных выплатах по сравнению с одиночным майнингом, но без чрезмерного контроля со стороны оператора пула в случае управляемых пулов.

Несмотря на снижение степени концентрации власти операторов пулов майнинга, P2Pool потенциально уязвим для «атак 51 %» на саму долевую цепочку. Более широкое внедрение P2Pool не решает проблему «атак 51 %» для самого Биткойна. Скорее, P2Pool в целом делает Биткойн более надежным

в качестве части диверсифицированной экосистемы майнинга. На момент написания данной книги P2Pool уже не используется, но новые протоколы, такие как Stratum v2, позволяют отдельным майнерам выбирать транзакции для включения в свои блоки.

Атаки на хешрейт

Механизм консенсуса Биткойна как минимум теоретически уязвим для атак майнеров (или пулов), которые пытаются использовать мощность своего хеширования для нечестных или разрушительных целей. Как было показано выше, механизм консенсуса зависит от честности большинства майнеров, работающих в своих интересах. Однако если майнеру или группе майнеров удастся получить серьезную долю мощности майнинга, они могут совершить атаку на механизм консенсуса и тем самым нарушить безопасность и жизнеспособность сети Биткойн.

Важно отметить, что атаки на хешрейт оказывают самое существенное влияние на будущий консенсус. Подтвержденные транзакции в лучшем блокчейне становятся с течением времени все более неизменяемыми. Хотя теоретически можно сделать форк на любой глубине, на практике вычислительные мощности, необходимые для очень глубокого форка, огромны, что делает замену старых блоков очень трудной задачей. Атаки на хешрейт также не затрагивают безопасность секретных ключей и алгоритмов подписи.

Один из сценариев атаки на механизм консенсуса называется *атакой большинства* (*majority attack*), или «*атакой 51 %*» (*51 % attack*). В этом сценарии группа майнеров, контролирующая большинство хеширующих мощностей сети (например, 51 %), вступает в сговор и атакует Биткойн. Имея возможность добывать большинство блоков, атакующие майнеры могут создавать преднамеренные «форки» в блокчейне и проводить двойные транзакции или осуществлять атаки типа «отказ в обслуживании» на определенные транзакции либо адреса. Атака типа «форк / двойная транзакция» предполагает, что злоумышленник делает ранее подтвержденные блоки недействительными путем форка ниже их и повторного перехода на другую цепочку. При достаточной мощности атакующий может аннулировать блоки на глубину до шести блоков и более, в результате чего транзакции, считавшиеся неизменными (шесть подтверждений), будут признаны недействительными. Обратите внимание, что двойное расходование может быть осуществлено только в отношении собственных транзакций злоумышленника, для которых он может создать действительную подпись. Двойное расходование собственных транзакций может быть выгодным, если признание транзакции недействительной позволяет атакующему провести необратимый обмен валюты или получить товар без оплаты.

Рассмотрим практический пример «атаки 51%». В первой главе рассматривалась транзакция между Алисой и Бобом. Продавец Боб готов принять платеж, не дожидаясь подтверждения (майнинга в блоке), потому что риск двойного расходования средств на небольшой товар невелик по сравнению с удобством быстрого обслуживания клиента. Это похоже на практику кофеен, которые принимают платежи по кредитным картам без подписи при сумме менее $25,

поскольку риск возврата средств с кредитной карты невелик, а затраты на задержку транзакции для получения подписи сравнительно больше. Напротив, при продаже более дорогого товара за биткойны возникает вероятность проведения атаки с двойным расходованием, когда покупатель передает конкурирующую транзакцию с расходованием тех же самых средств (UTXO) и отменяет выплату продавцу. «Атака 51 %» позволяет злоумышленникам дважды провести собственную транзакцию в новой цепочке, тем самым отменяя соответствующую транзакцию в старой цепочке.

В нашем примере злоумышленница Мэллори приходит в галерею Кэрол и покупает набор прекрасных картин, изображающих Сатоши Накамото в образе Прометея. Кэрол продает картины Мэллори за $250 000 в биткойнах. Вместо ожидания шести или более блоков для подтверждения сделки Кэрол упаковывает и передает картины Мэллори после единственного подтверждения. Мэллори работает с сообщником Полом, управляющим крупным пулом майнинга, и тот начинает атаку сразу после включения транзакции Мэллори в блокчейн. Пол направляет пул майнинга на повторный майнинг блока той же высоты, что и блок с транзакцией Мэллори, заменяя платеж от Мэллори в пользу Кэрол транзакцией с двойным расходованием того же входа, что и платеж Мэллори. Транзакция с двойным расходованием расходует тот же UTXO и возвращает его на кошелек Мэллори вместо выплаты Кэрол, по сути позволяя Мэллори оставить биткойны себе. Затем Пол дает команду пулу майнинга добыть дополнительный блок, чтобы цепочка с транзакцией двойного расходования стала длиннее исходной цепочки (что приводит к форку под блоком с транзакцией Мэллори). Когда форк блокчейна разрешается в пользу новой (более длинной) цепочки, транзакция с двойным расходованием заменяет первоначальный платеж Кэрол. Теперь у Кэрол не хватает трех картин, и она также не получила оплату. В течение всей этой операции участники пула майнинга Пола могут пребывать в полном неведении относительно попыток совершить двойное расходование, поскольку они занимаются майнингом на автоматических установках и не могут отслеживать каждую транзакцию либо блок.

Для защиты от подобных атак продавцам дорогостоящих товаров следует ждать не менее шести подтверждений, прежде чем отдать товар покупателю. Иногда может быть оправданным ожидание более шести подтверждений. В качестве альтернативы продавцу следует использовать целевой условно-блокированный (эскроу) счет с несколькими подписями, но при этом опять же нужно дождаться нескольких подтверждений после пополнения счета эскроу. Чем больше пройдет подтверждений, тем сложнее будет признать транзакцию недействительной из-за реорганизации блокчейна. Для дорогостоящих товаров оплата биткойнами будет удобной и эффективной, даже если покупателю придется ждать доставки 24 часа, а это соответствует примерно 144 подтверждениям.

Помимо атаки с двойным расходованием, возможен и другой сценарий атаки на консенсус – отказ в обслуживании конкретным участникам (конкретным адресам биткойнов). Злоумышленник с большим количеством мощностей для майнинга может осуществлять цензуру транзакций. Если они включены в блок, добытый другим майнером, атакующий может намеренно создать форк и заново майнить этот блок, снова исключая конкретные транзакции. Такой

тип атаки может привести к устойчивому отказу в обслуживании определенного адреса или набора адресов до тех пор, пока злоумышленник контролирует большинство мощностей для майнинга.

Несмотря на название, сценарий «атаки 51 %» на самом деле не требует 51 % мощности хеширования. На практике такая атака может быть предпринята и с меньшим процентом мощностей. Порог в 51 % – это только уровень, при котором такая атака почти гарантированно будет успешной. Атака по хешрейту – это, по сути, перетягивание каната за следующий блок, и более сильная группа, скорее всего, победит. При меньшей мощности хеширования вероятность успеха снижается, поскольку другие майнеры контролируют создание некоторых блоков с помощью своей «честной» мощности майнинга. Один из вариантов: чем больше мощность хеширования у злоумышленника, тем более продолжительный форк он может намеренно создать и тем больше блоков в недавнем прошлом он может признать недействительными, или же контролировать больше блоков в будущем. Группы исследователей в области безопасности, опираясь на статистическое моделирование, утверждают, что различные типы атак на хешрейт возможны при использовании всего 30 % мощности хеширования.

Централизация управления, вызванная появлением пулов для майнинга, влечет за собой угрозу коммерческих атак со стороны оператора пула. В управляемом пуле его оператор контролирует создание блоков-кандидатов, а также определяет входящие в них транзакции. Это дает оператору пула возможность исключать транзакции или вводить транзакции с двойным расходованием. Если такое злоупотребление властью происходит в ограниченном и незаметном режиме, оператор пула может извлекать прибыль из атак на хешрейт, оставаясь при этом незамеченным.

Впрочем, далеко не все атакующие руководствуются соображениями выгоды. Один из возможных сценариев атаки предполагает, что злоумышленник намерен нарушить работу сети Биткойн, не имея возможности извлечь из этого выгоду. Вредоносная атака, направленная на разрушение Биткойна, потребует огромных инвестиций и тайного планирования, но может быть осуществлена хорошо финансируемым злоумышленником, скорее всего, спонсируемым государством. Как вариант хорошо финансируемый злоумышленник может атаковать Биткойн, одновременно накапливая оборудование для майнинга, компрометируя операторов пулов и атакуя другие пулы с помощью отказа в обслуживании. Все эти сценарии теоретически возможны.

Несомненно, серьезная атака на хешрейт в краткосрочной перспективе подорвет доверие к Биткойну и, возможно, вызовет значительное падение его цены. Однако сеть и программное обеспечение Биткойна постоянно развиваются, поэтому на подобные атаки сообщество Биткойна всегда найдет ответные меры.

Изменение правил консенсуса

Правила консенсуса определяют подлинность транзакций и блоков. Эти правила являются основой для взаимодействия между всеми узлами Биткойна и отвечают за приведение всех локальных вариантов к единому согласованному блокчейну во всей сети.

Хотя правила консенсуса неизменны в краткосрочной перспективе и должны быть согласованы между всеми узлами, они не являются неизменными в долгосрочной перспективе. Для эволюции и развития системы Биткойн правила могут время от времени меняться, добавляя новые функции, улучшения или исправляя ошибки. Но в отличие от традиционного создания программ обновление системы консенсуса намного сложнее и требует координации между всеми участниками.

Хард-форки

В разделе «Сборка и выбор цепочек блоков» был рассмотрен процесс кратковременного разветвления сети Биткойн, когда две части сети в течение некоторого времени следуют по двум разным ветвям блокчейна. Этот процесс происходит естественным образом, как часть нормальной работы сети, и после майнинга одного или нескольких блоков сеть сходится к общему блокчейну.

Существует и другой сценарий расхождения сети на две цепочки: изменение правил консенсуса. Этот тип форка называется *хард-форком* (*hard fork*, буквально «*жесткая развилка*»), поскольку после него сеть не может вернуться к единой цепочке. Вместо этого две цепочки могут развиваться независимо друг от друга. Хард-форки происходят в тех случаях, когда часть сети работает по другому набору правил консенсуса, нежели остальная часть. Это может произойти из-за ошибки или намеренного нарушения правил консенсуса.

Хард-форки могут быть использованы для изменения правил консенсуса, но они требуют координации между всеми участниками системы. Узлы, не перешедшие на новые правила консенсуса, не могут участвовать в механизме консенсуса и в момент хард-форка вытесняются в отдельную цепочку. Таким образом, вносимые хард-форком изменения можно считать «несовместимыми в будущем», поскольку необновленные системы больше не смогут обрабатывать блоки из-за новых правил консенсуса.

Рассмотрим принцип работы хард-форка на конкретном примере.

На рис. 12.2 показан блокчейн с двумя форками. На высоте блока 4 происходит форк одного блока. Это тот тип спонтанной развилки, который был рассмотрен в разделе «Сборка и выбор цепочек блоков». После майнинга блока 5 сеть сходится к одной цепочке, и форк устраняется.

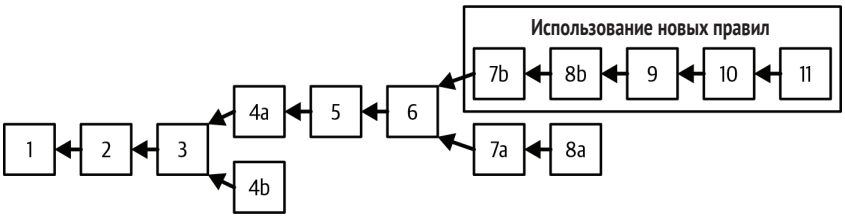


Рис. 12.2 ❖ Блокчейн с форками

Однако позже, на высоте блока 6, выходит новая реализация клиента с изменением правил консенсуса. Начиная с высоты блока 7 майнеры, использу-

ющие эту новую реализацию, будут принимать новый тип биткойна; назовем его «фукойн» (foocoin). Сразу после этого узел с новой реализацией создает транзакцию с фукойном, а майнер с обновленным программным обеспечением добывает блок 7b с этой транзакцией.

Любой узел или майнер, не обновивший программу для подтверждения фукойна, теперь не сможет обработать блок 7b. С их точки зрения, и транзакция с фукойном, и блок 7b с этой транзакцией являются недействительными, поскольку они оценивают их по старым правилам консенсуса. Эти узлы отклонят транзакцию и блок, а также не будут их распространять. Майнеры, использующие старые правила, не примут блок 7b и продолжат добывать блок-кандидат, родителем которого является блок 6. В действительности майнеры, использующие старые правила, могут даже не получить блок 7b, если все узлы, к которым они подключены, также подчиняются старым правилам и потому не рассылают этот блок. В конечном итоге они смогут добыть блок 7a, который действителен по старым правилам и не содержит никаких транзакций с фукойнами.

С этого момента две цепочки продолжают расходиться. Майнеры цепочки «b» продолжают принимать и майнить транзакции с фукойнами, в то время как майнеры цепочки «a» продолжают игнорировать эти транзакции. Даже если блок 8b не содержит транзакций с фукойнами, майнеры цепочки «a» не смогут его обработать. Для них он кажется недействительным блоком, поскольку его родитель «7b» не распознается как действительный.

Хард-форки: программы, сеть, майнинг и цепочка

Для разработчиков программ термин «форк» имеет другое значение, добавляя дополнительную неразбериху к термину «хард-форк». В программном обеспечении с открытым исходным кодом форк происходит по решению группы разработчиков следовать другому плану развития программного обеспечения и начать конкурирующую реализацию открытого проекта. Ранее уже обсуждались два обстоятельства, которые могут привести к хард-форку: ошибка в правилах консенсуса и преднамеренное изменение этих правил. В случае преднамеренного изменения правил консенсуса хард-форку предшествует программный форк. Однако для возникновения этого типа хард-форка необходимо разработать, принять и запустить новую программную реализацию правил консенсуса.

Примерами программных форков, которые пытались изменить правила консенсуса, являются Bitcoin XT и Bitcoin Classic. Однако ни одна из этих программ не привела к хард-форку. Хотя программный форк является необходимым условием, сам по себе он не является достаточным для возникновения хард-форка. Чтобы произошел хард-форк, необходимо, чтобы конкурирующая реализация была принята и новые правила были активированы майнерами, кошельками и узлами-посредниками. И наоборот, существует множество альтернативных реализаций Bitcoin Core и даже программных форков, которые не меняют правила консенсуса и, если не произойдет ошибка, могут сосуществовать в сети и взаимодействовать друг с другом, не вызывая хард-форка.

Правила консенсуса могут различаться очевидными и явными способами, например при проверке транзакций или блоков. Правила могут отличаться

и более тонко, в реализации правил консенсуса применительно к скриптам Биткойна или криптографическим примитивам, таким как цифровые подписи. Наконец, правила консенсуса могут отличаться непредвиденным образом из-за неявных ограничений консенсуса, налагаемых системными ограничениями или особенностями реализации. Примером последнего может служить непредвиденный хард-форк во время обновления Bitкойн Core 0.7 до 0.8, который был вызван ограничением в реализации Berkeley DB, используемой для хранения блоков.

Концептуально можно представить, что хард-форк развивается в четыре этапа: программный форк, сетевой форк, майнинг-форк и форк цепочки. Процесс начинается после того, как разработчики создают альтернативную реализацию клиента с измененными правилами консенсуса.

Когда эта форк-реализация будет развернута в сети, определенный процент майнеров, пользователей кошельков и промежуточных узлов могут принять и запустить ее. Сначала произойдет форк сети. Узлы с исходной реализацией правил консенсуса будут отвергать любые транзакции и блоки, созданные по новым правилам. Более того, узлы, придерживающиеся первоначальных правил консенсуса, могут отсоединиться от узлов, которые отправляют им эти недействительные транзакции и блоки. В результате сеть может разделиться на две части: старые узлы останутся соединенными только со старыми, а новые узлы будут соединены только с новыми. Один-единственный блок, основанный на новых правилах, разойдется по сети и приведет к разделению на две сети.

Новые майнеры могут майнить поверх нового блока, в то время как старые майнеры будут майнить отдельную цепочку, основанную на старых правилах. Разделенная сеть приведет к тому, что майнеры, работающие по разным правилам консенсуса, вряд ли получат блоки друг друга, поскольку они подключены к двум разным сетям.

Расхождение майнеров и сложности

Как только майнеры расходятся для майнинга двух разных цепочек, мощности хеширования будут также распределены между этими двумя направлениями. При этом мощности майнинга могут быть разделены между ними в любой пропорции. Новым правилам может следовать как меньшинство, так и подавляющее большинство майнеров.

Предположим, что разделение составит, например, 80–20 %, и большинство майнеров будут использовать новые правила консенсуса. Также представим, что форк происходит сразу после периода ретаргетинга.

Две цепочки унаследуют трудности периода ретаргетинга. В новых правилах консенсуса будет задействовано 80 % ранее доступных мощностей майнинга. С точки зрения этой цепочки, мощность майнинга внезапно снизилась на 20 % по сравнению с предыдущим периодом. Блоки будут появляться в среднем каждые 12,5 минуты, что представляет собой 20%-ное снижение мощности майнинга, доступной для продления этой цепочки. Такая скорость выдачи блоков будет продолжаться (при отсутствии каких-либо изменений в мощности хеширования) до тех пор, пока не будет добыто 2016 блоков, что займет примерно 25

200 минут (при 12,5 минуты на блок), или 17,5 дня. Через 17,5 дня произойдет ретаргетинг, и сложность будет скорректирована (уменьшена на 20 %) для достижения 10-минутного интервала между блоками с учетом уменьшения количества мощностей хеширования в этой цепочке.

Цепочка меньшинства, добывающая по старым правилам лишь 20 % мощностей хеширования, столкнется с гораздо более сложной задачей. Теперь на этой цепочке блоки будут добываться в среднем каждые 50 минут. Сложность не будет регулироваться в течение 2016 блоков, на добычу которых уйдет 100 800 минут, или около 10 недель. Если исходить из фиксированного расхода ресурсов на добычу одного блока, это также приведет к сокращению объема транзакций в 5 раз, поскольку для записи транзакций будет доступно меньшее количество блоков в час.

Спорный механизм хард-форков

Сегодняшний день – это новая эпоха в истории разработки программного обеспечения для децентрализованного консенсуса. Как и другие инновации в разработке, они изменили как методы создания, так и сами программные продукты. Появились новые стратегии, инструменты и сообщества. Разработка программ для достижения консенсуса также представляет собой новый рубеж в компьютерной науке. В результате дискуссий, экспериментов и испытаний, связанных с разработкой Биткойна, появляются новые инструменты, практики, технологии и профессиональные группы.

Хард-форки считают опасным явлением, поскольку они вынуждают меньшинство либо обновляться, либо оставаться в цепочке меньшинства. Риск разделения всей платформы на две конкурирующие системы рассматривается многими как неприемлемый. В связи с этим многие разработчики неохотно используют механизм хард-форков для внедрения обновлений правил консенсуса, если это не находит практически единодушной поддержки со стороны всей сети. Любые предложения по хард-форку, не имеющие практически единогласной поддержки, считаются чересчур спорными для их реализации без риска разделения всей системы.

Уже сейчас можно наблюдать появление новых методик, направленных на устранение рисков возникновения хард-форков. В следующем разделе будут рассмотрены софт-форки, а также методы подачи сигналов и активации модификаций консенсуса.

Софт-форки

Далеко не все изменения правил консенсуса приводят к хард-форку. Только изменения без обратной совместимости с прежними правилами консенсуса приводят к форку. Если изменения реализованы так, что клиент без внесенных поправок по-прежнему считает транзакцию или блок действительными в соответствии с предыдущими правилами, эти изменения могут произойти без форка.

Термин *софт-форк* (soft fork, буквально «мягкая развилка») был введен для обозначения отличия этого метода обновления от хард-форка. На практике софт-форк – это вовсе не форк. Скорее, это изменение правил консенсуса для

дальнейшего сохранения совместимости, которое позволяет необновленным клиентам работать с новыми правилами в консенсусе.

Один из не самых очевидных аспектов софт-форков заключается в том, что обновления с их помощью можно использовать только для ограничения правил консенсуса, но не для их расширения. Для сохранения совместимости все транзакции и блоки, созданные по новым правилам, должны быть действительны и по старым правилам, но не наоборот. Новые правила могут только ограничивать их правомерность; в противном случае – когда они будут отклонены по старым правилам – это спровоцирует возникновение хард-форка.

Софт-форки могут быть реализованы несколькими способами – термин не определяет конкретный подход, а, скорее, описывает совокупность методов с одной общей чертой: они не требуют обновления всех узлов и не принуждают необновленные узлы выходить из консенсуса.

В Биткойне реализованы два софт-форка на основе новой интерпретации операционных кодов NOP. В скрипте Биткойна для будущего использования были зарезервированы 10 операций с NOP1 по NOP10. Согласно правилам консенсуса, наличие этих кодов в скрипте интерпретируется как оператор с нулевым потенциалом, то есть они не оказывают никакого влияния. Исполнение скрипта продолжается после кода NOP, словно его и не было.

Таким образом, софт-форк может изменить семантику кода NOP, придав ему новое значение. Например, BIP65 (`CHECKLOCKTIMEVERIFY`) по-новому интерпретирует код NOP2. Клиенты с реализацией BIP65 интерпретируют NOP2 как `OP_CHECKLOCKTIMEVERIFY` и накладывают на UTXO с этим кодом в своих скриптах блокировки правило абсолютного консенсуса по времени блокировки. Это изменение является софт-форком, поскольку транзакция, действительная в соответствии с BIP65, также действительна на любом клиенте, не применяющем (не знающем) BIP65. Для старых клиентов скрипт содержит код NOP, который просто игнорируется.

Критика софт-форков

Софт-форки на основе операций NOP являются сравнительно безопасными. Коды NOP были включены в скрипт Биткойна с очевидной целью обеспечить возможность обновления без сбоев. Однако многие разработчики обеспокоены тем, что другие методы обновления софт-форков приводят к неприемлемым компромиссам. К числу распространенных критических замечаний в адрес софтфорков относятся следующие.

Техническая задолженность
: Поскольку софт-форки сложнее в техническом плане, чем хард-форки, они приводят к появлению *технической задолженности* (*technical debt*). Этому термину соответствует увеличение будущих затрат на поддержку кода из-за принятых в прошлом компромиссов при разработке. Сложность кода, в свою очередь, увеличивает вероятность появления ошибок и уязвимостей в системе безопасности.

Ослабление валидации
: Немодифицированные клиенты воспринимают транзакции как действительные без оценки измененных правил консенсуса. По сути, немодифици-

рованные клиенты не проверяют транзакции с применением всего спектра правил консенсуса, поскольку они не видят новых правил. Это относится к обновлениям на основе NOP, а также к другим обновлениям софт-форка.

Необратимые обновления

Поскольку софт-форки создают транзакции с дополнительными ограничениями консенсуса, с практической точки зрения они становятся необратимыми обновлениями (irreversible upgrade). Если обновление софт-форка будет отменено после его активации, любые транзакции, созданные по новым правилам, могут привести к потере средств по старым правилам. Например, если транзакция CLTV оценивается по старым правилам, то у нее нет ограничения по времени, и она может быть потрачена в любой момент. Поэтому многие эксперты считают, что неудачный софт-форк, который пришлось отменять из-за ошибки, почти наверняка приведет к потере средств.

Сигналы софт-форка по версии блока

Поскольку софт-форки позволяют немодифицированным клиентам продолжать работу в рамках консенсуса, одним из механизмов «активации» софт-форка является сигнал майнеров о том, что они готовы и намерены соблюдать новые правила консенсуса. Если все майнеры будут соблюдать новые правила, то не будет риска, что немодифицированные узлы примут блок, который модернизированные узлы отвергнут. Этот механизм был введен в BIP34.

BIP34: сигналы и активация

Протокол BIP34 задействовал поле версии блока для отправки майнерам сигнала о готовности к определенному изменению правил консенсуса. До BIP34 версия блока устанавливалась на «1» *по соглашению*, не принуждаемому *консенсусом*.

BIP34 определил изменение правила консенсуса, которое обязывало содержать в поле coinbase (input) транзакции coinbase высоту блока. До BIP34 поле coinbase могло содержать любые произвольные данные, выбранные майнерами. После активации BIP34 действительные блоки должны были содержать определенную высоту блока в начале coinbase и быть идентифицированы номером версии блока, большим или равным «2».

Чтобы подать сигнал о своей готовности выполнить правила BIP34, майнеры устанавливают версию блока «2», а не «1». При этом блоки версии «1» не сразу становятся недействительными. Но после активации блоки версии «1» становятся недействительными, а все блоки версии «2» для признания их действительными должны содержать высоту блока в coinbase.

BIP34 определил двухэтапный механизм активации, основанный на переходе через окно в 1000 блоков. Майнер должен был подать сигнал о своей готовности к BIP34, создавая блоки с «2» в качестве номера версии. Строго говоря, эти блоки еще не должны были соответствовать новому правилу консенсуса, предусматривающему включение высоты блока в транзакцию coinbase, поскольку правило консенсуса еще не было активировано. Правила консенсуса активируются в два этапа:

- если 75 % (750 из последних 1000 блоков) помечены версией «2», блоки версии «2» должны содержать высоту блока в транзакции coinbase, иначе они отклоняются как недействительные. Блоки версии «1» по-прежнему принимаются сетью и не должны содержать высоту блока. Старые и новые правила консенсуса сосуществуют в течение этого периода;
- если 95 % (950 из последних 1000 блоков) имеют версию «2», блоки версии «1» больше не считаются действительными. Блоки версии «2» считаются действительными, только если они содержат высоту блока в coinbase (в соответствии с предыдущим пороговым значением). После этого все блоки должны соответствовать новым правилам консенсуса, а все действительные блоки должны содержать высоту блока в транзакции coinbase.

После успешной подачи сигнала и активации по правилам BIP34 этот механизм был еще дважды использован для активации софт-форков:

- BIP66 https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki (Strict DER Encoding of Signatures) был активирован сигналом как в BIP34 с версией блока «3»;
- BIP65 https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki `CHECKLOCKTIMEVERIFY` был активирован сигналом как в BIP34 с версией блока «4».

После активации BIP65 механизм подачи сигналов и активации BIP34 был упразднен и заменен на описанный далее алгоритм подачи сигналов BIP9.

BIP9: сигналы и активация

Механизм в протоколах BIP34, BIP66 и BIP65 успешно активировал три софт-форка, однако был заменен из-за ряда ограничений.

- Из-за целочисленного значения версии блока за один раз можно было активировать только один софт-форк, поэтому требовались координация между предложениями софт-форков и согласование их приоритетности и последовательности.
- Более того, поскольку версия блока увеличивалась, механизм не предлагал прямого способа отклонить изменение и затем предложить другое. Если бы старые клиенты все еще работали, они могли бы принять сигнал о новом изменении за сигнал о ранее отклоненном изменении.
- Каждое новое изменение необратимо уменьшало доступные версии блоков для будущих изменений.

Для решения этих проблем и повышения скорости и простоты реализации будущих изменений был предложен BIP9.

Протокол BIP9 интерпретирует версию блока как битовое поле, а не целое число. Поскольку версия блока изначально использовалась как целое число для версий с 1 по 4, для использования в качестве битового поля остается только 29 бит. Таким образом, 29 бит могут быть использованы для независимой и одновременной подачи сигнала готовности по 29 различным предложениям.

В BIP9 также установлено максимальное время для подачи сигнала и активации. Таким образом, майнерам не придется подавать сигналы до бесконеч-

ности. Если предложение не активировано в течение периода TIMEOUT (определенного в предложении), оно считается отклоненным. Это предложение может быть повторно отправлено с другим битом для подачи сигнала, что продлит период активации.

Кроме того, после истечения TIMEOUT и активации или отклонения предложения бит сигнала может быть повторно использован для другого предложения без какой-либо путаницы. Таким образом, параллельно можно подать сигнал о 29 изменениях. По истечении TIMEOUT биты могут быть «повторно использованы» для предложения новых изменений.

Хотя биты сигналов можно использовать повторно или многократно, при условии что период голосования не пересекается, авторы BIP9 рекомендуют использовать биты повторно только в случае крайней необходимости; непредвиденное изменение может быть вызвано ошибками в старом программном обеспечении. Одним словом, не стоит ожидать повторного использования до тех пор, пока все 29 бит не будут использованы хотя бы единожды.

Предлагаемые изменения идентифицируются структурой данных со следующими полями:

имя
: Краткое описание для отличия предложений друг от друга. Чаще всего это BIP, описывающий предложение как «bipN», где N – номер BIP;

бит
: От 0 до 28, это бит в версии используемого майнерами блока, который служит сигналом для одобрения данного предложения;

время начала (starttime)
: Время (основанное на MTP) начала подачи сигнала, после которого значение бита интерпретируется как сигнал о готовности предложения;

время завершения (endtime)
: Время (на основе MTP), по истечении которого изменение считается отклоненным, если оно не достигло порога активации.

В отличие от BIP34, протокол BIP9 считает сигналы активации в целых интервалах, основываясь на периоде ретаргетинга сложности в 2016 блоков. Для каждого периода ретаргетинга, если сумма блоков, сигнализирующих о предложении, превышает 95 % (1916 из 2016), предложение будет активировано на один период ретаргетинга позже.

BIP9 предоставляет диаграмму состояния предложения, которая показывает различные этапы и переходы для предложения, как показано на рис. 12.3.

Предложения переходят в состояние DEFINED, как только их параметры становятся известны (определены) в программном обеспечении Биткойна. Для блоков с MTP после времени запуска состояние предложения переходит в STARTED. Если порог голосования превышен в течение периода ретаргетинга и при этом не было превышено время ожидания, состояние предложения переходит в LOCKED_IN. Через один период ретаргетинга предложение переходит в состояние ACTIVE. После перехода в состояние ACTIVE предложения остаются в нем на-

всегда. Если время ожидания истекает до достижения порога голосования, состояние предложения меняется на FAILED. Это означает, что предложение было отклонено. Отклоненные предложения остаются в этом состоянии навсегда.

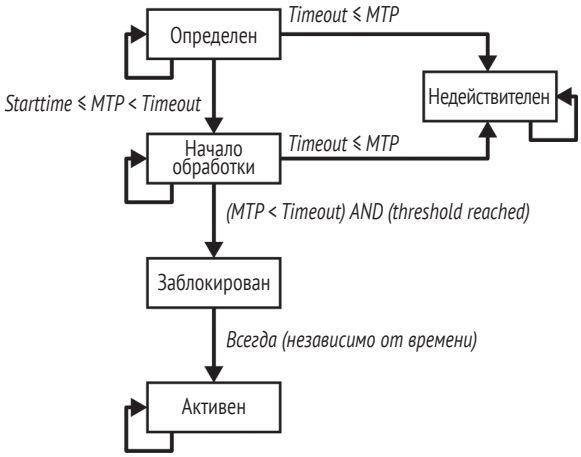


Рис. 12.3 ❖ Диаграмма переходов состояний BIP9

Впервые BIP9 был реализован для активации CHECKSEQUENCEVERIFY и связанных с ним BIP 68, BIP 112 и BIP 113. Предложение под названием «csv» было успешно активировано в июле 2016 года.

Этот стандарт определен в BIP9 https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki (биты версии с тайм-аутом и задержкой).

BIP8: обязательная блокировка с досрочной активацией

После успешного применения BIP9 для софт-форка CSV в следующей реализации изменения консенсуса с помощью софт-форка также попытались использовать его для принудительной активации со стороны майнеров. Однако некоторые участники выступили против этого предложения софт-форка под названием segwit, и лишь немногие майнеры в течение нескольких месяцев подали сигнал о готовности к его внедрению.

Позже выяснилось, что часть майнеров, в особенности несогласные, могли использовать оборудование, которое давало им скрытое преимущество над другими майнерами благодаря функции под названием «*скрытый ASICBoost*» (*covert ASICBoost*). Внедрение segwit непреднамеренно препятствовало возможности использования этого скрытого ASICBoost. В случае активации segwit майнеры, использующие его, теряли свое скрытое преимущество.

После выявления сообществом этого конфликта интересов некоторые пользователи приняли решение использовать свои полномочия, чтобы не принимать блоки от майнеров, если эти блоки не соответствуют определенным правилам. В конечном итоге пользователи хотели получить новые правила с учетом segwit. Они решили приумножить свои усилия за счет большого количества узлов, которые были готовы применить правила segwit, при условии

что достаточное количество майнеров подадут сигнал о готовности к этому. Анонимный разработчик предложил BIP148, который обязывал любой узел, реализующий его, отклонять все блоки без сигнала о готовности к segwit, начиная с определенной даты и до тех пор, пока segwit не будет активирован.

Хотя лишь ограниченное число пользователей действительно запустили код BIP148, многие другие были согласны с этим мнением и, возможно, уже готовились к принятию BIP148. За несколько дней до вступления BIP148 в силу почти все майнеры начали подавать сигналы о своей готовности выполнять правила segwit. Примерно через две недели segwit достиг порога блокировки и приблизительно через две недели после этого был активирован.

Многие пользователи посчитали недостатком BIP9 возможность майнеров предотвратить успешную попытку активации, не подавая сигналов в течение года. Им нужен был механизм, который гарантировал бы активацию софт-форка к определенной высоте блока, но при этом позволял бы майнерам подавать сигнал о готовности заблокировать его еще раньше.

Для этого был разработан протокол BIP8, который похож на BIP9, за исключением определения периода `MUST_SIGNAL`, в течение которого майнеры должны подать сигнал о готовности к реализации предложения о софт-форке.

В 2021 году было опубликовано программное обеспечение, использующее BIP8 для активации предложения taproot, и есть свидетельства того, что по крайней мере небольшое количество пользователей запустили это ПО. Некоторые из них также утверждают, что их готовность использовать BIP8 для принуждения майнеров к активации taproot стала причиной ее конечной активации. Они утверждают, что если бы taproot не был активирован быстро, другие пользователи тоже стали бы применять BIP8. К сожалению, проверить это невозможно, поэтому нельзя однозначно говорить о степени влияния BIP8 на активацию taproot.

Скорый суд: быстрый провал или финальный успех

Хотя сама идея BIP9, несмотря на широкую поддержку предложения, не стала причиной активации segwit, многим разработчикам протоколов было неясно, что BIP9 потерпел неудачу. Как уже говорилось, неспособность майнеров изначально подать сигнал о поддержке segwit могла быть в значительной степени результатом единовременного конфликта интересов, которого в будущем не возникнет. Кому-то казалось, что BIP9 стоит попробовать еще раз. Другие не соглашались и предпочитали использовать BIP8.

После нескольких месяцев обсуждений между наиболее заинтересованными в конкретных идеях активации участниками был предложен компромисс для активации taproot. Модифицированная версия BIP9 давала майнерам очень короткий промежуток времени для подачи сигнала о готовности применить правила taproot. Если бы подача сигнала не увенчалась успехом, можно было бы использовать другой механизм активации (или, возможно, отказаться от этой идеи). Если сигнал был бы успешным, принудительное исполнение правил началось бы примерно через шесть месяцев на определенной высоте блока. Один из тех, кто помогал продвигать этот механизм, назвал его «*ускоренным испытанием*» (*speedy trial*, буквально «*скорый суд*»).

Ускоренное рассмотрение активации было опробовано, майнеры быстро подали сигнал о готовности соблюдать правила taproot, и примерно через полгода taproot был успешно активирован. Для сторонников ускоренного процесса активации это был несомненный успех. Другие же были разочарованы отказом от использования BIP8.

Пока неясно, будет ли ускоренный процесс активации вновь использован для попыток внедрения софт-форка в будущем.

Разработка ПО для консенсуса

Программное обеспечение для консенсуса продолжает развиваться, и сейчас ведется много дискуссий о различных механизмах изменения правил консенсуса. По своей природе Биткойн устанавливает очень высокую планку для внесения изменений с помощью координации и консенсуса. Будучи децентрализованной системой, он не имеет «власти», которая могла бы навязать свою волю участникам сети. Полномочия распределены между многочисленными участниками, такими как майнеры, разработчики протоколов, создатели кошельков, биржи, продавцы и конечные пользователи. Решения не могут быть приняты в одностороннем порядке ни одной из этих групп. Например, майнеры могут цензурировать транзакции простым большинством голосов (51 %), однако они ограничены согласием других участников. Если они действуют в одностороннем порядке, остальные участники могут отказаться принимать их блоки, сохраняя экономическую активность в цепочке меньшинства. Без экономической активности (транзакций, продавцов, кошельков, бирж) майнеры будут добывать ничего не стоящую валюту с пустыми блоками. Такое распределение власти означает, что все участники должны координировать свои действия, иначе никаких изменений не произойдет. Статус-кво – это стабильное состояние этой системы, где возможны лишь некоторые изменения при наличии убедительного консенсуса со стороны подавляющего большинства. Планка в 95 % для софт-форков как раз отражает эту реальность.

Важно понимать, что идеального решения для достижения консенсуса не существует. Как хард-форки, так и софт-форки предполагают компромиссы. Для некоторых типов изменений лучшим выбором могут быть софт-форки, для других – хард-форки. Не существует безупречного выбора; оба варианта несут в себе риски. Неизменной характеристикой процесса разработки программного обеспечения на основе консенсуса является сложность изменений, а консенсус вынуждает идти на компромисс.

Некоторые считают это слабостью систем консенсуса. Со временем можно прийти к выводу, что это самая сильная сторона системы.

На данном этапе книги разговор о самой системе Биткойн закончен. Остались только вопросы о программном обеспечении, инструментах и других протоколах, созданных на основе Биткойна.

Глава 13

Безопасность Биткойна

Обеспечить безопасность ваших биткойнов очень сложно, поскольку они совсем не похожи на баланс банковского счета. Ваши биткойны во многом подобны цифровым деньгам или золоту. Наверняка вы слышали выражение «Владение – это девять десятых права» («Possession is nine-tenths of the law»). Что ж, в Биткойне владение – это десять десятых права. Владение ключами для расходования определенных биткойнов эквивалентно владению наличными или куском драгоценного металла. Их могут украсть. Можно потерять их, неудачно распорядиться ими или случайно передать кому-то не ту сумму. В каждом из этих случаев у пользователей нет никаких средств защиты в рамках протокола, как если бы они уронили наличные на тротуар в общественном месте.

Однако система Биткойн обладает возможностями, которых нет у наличных денег, золота и банковских счетов. Биткойн-кошелек с вашими ключами можно резервировать, как любой файл. Его можно хранить в нескольких копиях и даже распечатать для резервного хранения на бумаге. Вы не можете «создать резервную копию» наличных, золота или банковских счетов. Биткойн значительно отличается от всего, что было до него, поэтому и о защите биткойнов нужно думать по-новому.

Принципы безопасности

Основным принципом Биткойна является децентрализация, и это имеет важные последствия для безопасности. Централизованная модель, такая как традиционный банк или платежная сеть, зависит от контроля доступа и проверок с целью не допустить проникновения злоумышленников в систему. В отличие от этого, децентрализованная система, такая как Биткойн, передает ответственность и контроль пользователям. Поскольку безопасность сети основана на независимой верификации, сеть может быть открытой, а для трафика Биткойна не требуется шифрование (хотя оно все же может быть полезным).

В традиционной платежной сети, такой как система кредитных карт, платеж является бессрочным, поскольку содержит частный идентификатор пользователя (номер кредитной карты). После первоначального списания средств лю-

бой человек с доступом к идентификатору может «вытянуть» средства и снимать их с владельца снова и снова. Таким образом, платежная сеть должна быть защищена сквозным шифрованием и гарантировать, что никакие посторонние лица или посредники не смогут скомпрометировать платежный трафик во время транзита или хранения (без движения). Если злоумышленник получит доступ к системе, он может скомпрометировать текущие транзакции и платежные идентификаторы, которые могут быть использованы для создания новых транзакций. Еще хуже, когда данные клиентов оказываются скомпрометированными. Эти клиенты подвергаются опасности кражи личных данных и должны принять меры для предотвращения махинаций с использованием скомпрометированных счетов.

Принцип работы Биткойна существенно отличается. Транзакция в биткойнах санкционирует только конкретное значение для конкретного получателя и не может быть подделана. Она не раскрывает никакой частной информации, например личности сторон, и не может быть использована для авторизации дополнительных платежей. По этой причине платежная сеть Биткойн не нуждается в шифровании или защите от перехвата. Фактически транзакции Биткойна можно передавать по открытому каналу связи, такому как незащищенный Wi-Fi или Bluetooth, без ущерба для безопасности.

Децентрализованная модель безопасности Биткойна наделяет пользователей огромной властью. Вместе с ней приходит и ответственность за сохранение секретности своих ключей. Для большинства пользователей это непростая задача, особенно на вычислительных устройствах общего назначения, таких как подключенные к интернету смартфоны или ноутбуки. И хотя децентрализованная модель Биткойна не допускает таких массовых компрометаций, как в случае с кредитными картами, многие пользователи не в состоянии должным образом защитить свои ключи и в результате один за другим становятся жертвами взлома.

Безопасная разработка систем Биткойн

Важнейшим фактором для разработчиков Биткойна является децентрализация. Большинство из них знакомы с централизованными моделями безопасности, и у некоторых может возникнуть соблазн применить эти модели к своим приложениям для Биткойна, что привело бы к плачевным результатам.

Безопасность Биткойна основывается на децентрализованном контроле над ключами и независимом подтверждении транзакций пользователями. Если при этом планируется задействовать все преимущества безопасности Биткойна, необходимо не выходить за рамки модели его безопасности. Проще говоря, не забирайте контроль над ключами у пользователей и не передавайте подтверждение транзакций на аутсорсинг.

Так, например, многие из ранних бирж Биткойна собирали все средства пользователей в один «горячий» кошелек и хранили ключи на едином сервере. Такая конструкция лишает пользователей контроля и централизует управление ключами в единой системе. Многие такие системы были взломаны, что привело к катастрофическим последствиям для их клиентов.

Если вы не готовы серьезно инвестировать в обеспечение операционной безопасности, многоуровневый контроль доступа и аудит (как это делают тра-

дициионные банки), вам следует очень хорошо подумать, прежде чем выводить средства за пределы децентрализованного механизма безопасности Биткойна. Даже если у вас есть средства и дисциплина для реализации надежной модели безопасности, такая конструкция просто повторяет хрупкую модель традиционных финансовых сетей, страдающих от краж личных данных, коррупции и растрат. Чтобы воспользоваться всеми возможностями уникальной децентрализованной модели безопасности Биткойна, стоит избегать соблазна централизованных архитектур, которые могут показаться вполне естественными, но в конечном итоге подрывают безопасность Биткойна.

Корень доверия

Традиционная архитектура безопасности основана на концепции под названием «*корень доверия*» (*root of trust*), которая представляет собой доверенное ядро в качестве основы для безопасности всей системы или приложения. Архитектура безопасности строится вокруг корня доверия в виде нескольких концентрических кругов, как слои в луковице, распространяя надежность от центра. Каждый слой основывается на более надежном внутреннем слое за счет элементов управления доступом, цифровых подписей, шифрования и других примитивов безопасности. С усложнением программных систем возрастает и вероятность наличия в них ошибок, повышающих степень уязвимости к угрозам безопасности. Как следствие чем сложнее программная система, тем труднее ее защитить. Концепция корня доверия гарантирует, что большая часть обеспечения надежности возлагается на наименее сложную часть системы, а значит, и на наименее уязвимые ее части, в то время как более сложное ПО располагается слоями вокруг нее. Эта архитектура безопасности повторяется в различных масштабах, вначале создавая корень доверия в аппаратном обеспечении одной системы, затем распространяя этот корень доверия через операционную систему на системные службы более высокого уровня и, наконец, на множество серверов, расположенных в виде концентрических кругов убывающего уровня надежности.

Архитектура безопасности Биткойна устроена совершенно иначе. Система консенсуса в Биткойне создает надежный, полностью децентрализованный блокчейн. В корректно подтвержденном блокчейне в качестве корня доверия используется блок генезиса, который выстраивает цепочку доверия вплоть до текущего блока. Системы Биткойн могут и должны использовать блокчейн в качестве корня доверия. При разработке сложного биткойн-приложения, состоящего из сервисов на различных системах, следует внимательно изучить всю архитектуру безопасности, чтобы убедиться, на чем именно основывается доверие. В конечном итоге единственное, чему следует явно доверять, – это полностью подтвержденный блокчейн. Если ваше приложение явно или неявно доверяет чему-либо, помимо блокчейна, это должно вызывать беспокойство из-за риска появления уязвимости. Хорошим методом оценки архитектуры безопасности вашего приложения является анализ каждого отдельного компонента и оценка гипотетического сценария, в котором этот компонент полностью скомпрометирован и находится под контролем злоумышленника. Возьмите поочередно каждый такой компонент вашего приложения и оцените

последствия для общей безопасности в случае его взлома. Если ваше приложение перестает быть безопасным при взломе компонентов, это говорит о неправильном распределении доверия к этим компонентам. Биткойн-приложение без слабых мест должно быть уязвимо только при компрометации механизма консенсуса Биткойна, что означает, что его корень доверия основан на самой сильной части архитектуры безопасности Биткойна.

Многочисленные примеры взлома бирж Биткойна наглядно подтверждают этот тезис, поскольку их архитектура и структура безопасности не выдерживают критики даже при поверхностном рассмотрении. Эти централизованные системы явно доверяли многочисленным компонентам вне блокчейна Биткойна, таким как горячие кошельки, централизованные базы данных, уязвимые ключи шифрования и другие подобные схемы.

Лучшие практики безопасности для пользователей

Человечество использует средства физической защиты на протяжении тысячелетий, в то время как его опыт в области цифровой безопасности составляет менее 50 лет. Современные операционные системы общего назначения недостаточно безопасны и не особенно подходят для хранения цифровых денег. Наши компьютеры постоянно подвергаются внешним угрозам через постоянно включенное интернет-подключение. На них работают тысячи приложений от сотен авторов, часто с неограниченным доступом к пользовательским файлам. Одна-единственная вредоносная программа из многих тысяч, установленных на вашем компьютере, может взломать клавиатуру и файлы, а также украсть все биткойны из приложений-кошельков. Уровень технической поддержки компьютера, необходимый для защиты от вирусов и троянов, не под силу многим пользователям, за исключением небольшого меньшинства.

Несмотря на десятилетия исследований и достижений в области информационной безопасности, цифровые активы по-прежнему крайне уязвимы для агрессивного злоумышленника. Даже наиболее защищенные и закрытые системы финансовых компаний, разведывательных служб и оборонных подрядчиков нередко подвергаются взлому. Биткойн создает цифровые активы, которые имеют внутреннюю ценность и могут быть украдены и переведены на новых владельцев мгновенно и безвозвратно. Это служит огромным стимулом для хакеров. До сих пор им приходилось превращать информацию о личности или учетные маркеры, например кредитные карты либо банковские счета, в ценные активы после их компрометации. Несмотря на сложности с обеспечением защиты и отмывания финансовых данных, количество краж постоянно увеличивается. Биткойн усугубляет эту проблему, поскольку он не нуждается в защите или отмывании: биткойны ценны сами по себе.

Биткойн также стимулирует к повышению уровня компьютерной безопасности. Если раньше риск взлома компьютера был неопределенным и неявным, то с Биткойном этот риск становится понятным и очевидным. Хранение биткойнов на компьютере фокусирует внимание пользователя на необходимости

повышения уровня защиты системы. Прямым результатом распространения и все более широкого использования биткойнов и других цифровых валют стала эскалация как хакерских технологий, так и решений по обеспечению безопасности. Проще говоря, у хакеров появилась очень привлекательная цель, а у пользователей – очевидный стимул защищаться.

В последние три года в результате распространения Биткойна мы стали свидетелями огромного количества инноваций в сфере информационной безопасности в виде аппаратного шифрования, устройств хранения ключей и аппаратной подписи, технологии мультиподписи и цифрового депонирования. В следующих разделах будут рассмотрены различные передовые методы обеспечения безопасности пользователей на практике.

Физическое хранение биткойнов

Для большинства пользователей физическая безопасность привычнее информационной, поэтому очень эффективным методом защиты биткойнов является перевод их в физическую форму. Ключи биткойнов и исходные тексты, используемые для их создания, представляют собой не что иное, как длинные числа. Это означает, что их можно хранить в физической форме, например напечатать на бумаге или выгравировать на металлической пластине. В этом случае защита ключей сводится к физическому сохранению распечатанной копии «семени» (seed) ключа. Распечатанная на бумаге копия ключа называется «бумажной резервной копией» (paper backup), и многие кошельки умеют ее создавать. Хранение биткойнов в автономном режиме называется «*холодным хранением*» (*cold storage*) и является одним из наиболее эффективных методов защиты. В системе холодного хранения ключи генерируются на автономном устройстве (никогда не подключающемся к интернету) и хранятся либо на бумаге, либо на цифровом носителе, например на USB-накопителе.

Аппаратные устройства подписи

В долгосрочной перспективе безопасность Биткойна может все чаще выражаться в использовании аппаратных устройств для подписи с защитой от взлома. В отличие от смартфона или настольного компьютера, аппаратное устройство для подписи биткойнов хранит только ключи и использует их для генерации подписей. Не имея программного обеспечения общего назначения, которое можно взломать, и обладая ограниченным числом интерфейсов, аппаратные устройства подписи могут обеспечить надежную защиту для неопытных пользователей. Возможно, именно такие устройства могут стать преобладающим методом хранения биткойнов.

Безопасность вашего доступа

Хотя большинство пользователей справедливо опасаются кражи своих биткойнов, существует гораздо более серьезный риск. Файлы данных теряются постоянно. Если они содержат ключи от биткойнов, потеря будет гораздо более

болезненной. Стремясь обезопасить свои биткойн-кошельки, пользователям стоит проявлять большую осторожность, чтобы не зайти слишком далеко и не потерять свои биткойны. В июле 2011 года известный информационно-просветительский проект по Биткойну потерял почти 7000 биткойнов. Пытаясь предотвратить кражу, его владельцы создали сложную систему зашифрованных резервных копий. В итоге ключи шифрования были случайно утеряны, из-за чего резервные копии оказались бесполезными, и они потеряли целое состояние. Как и в случае со спрятанными в пустыне деньгами, если слишком хорошо спрятать биткойны, то потом их можно и не найти.

Для расходования биткойнов вам может потребоваться не только резервная копия секретных ключей или семян BIP32, использованных для их создания. Это особенно актуально при работе с несколькими подписями или сложными скриптами. В большинстве скриптов выхода указываются реальные условия, которые должны быть выполнены для расходования биткойнов в этом выводе. Выполнить эти обязательства будет невозможно, если программа вашего кошелька не сможет сообщить о них сети. Коды восстановления кошелька обязательно должны содержать эти данные. Подробнее об этом см. в главе 5.

Диверсификация рисков

Стали бы вы хранить все свое состояние в виде наличности в своем кошельке? Большинство людей сочли бы это безрассудством, однако пользователи биткойнов очень часто держат все свои биткойны в единственном приложении-кошельке. Вместо этого им стоит распределить риск между несколькими разноплановыми биткойн-приложениями. Осмотрительные пользователи хранят лишь небольшую часть, возможно, менее 5 % своих биткойнов в онлайн- или мобильном кошельке в качестве «карманной мелочи». Остальные средства лучше разделить между несколькими различными способами хранения, такими как кошелек для настольных систем и офлайн (холодное хранение).

Мультиподпись и управление

Когда компании или частные лица хранят большие суммы биткойнов, им стоит подумать об использовании биткойн-адреса с несколькими подписями (мультиподписью – multisignature). Адреса с мультиподписью повышают безопасность средств, поскольку для проведения платежа требуется более одной подписи. Ключи подписи необходимо хранить в разных местах и под контролем разных людей. Например, в корпоративной среде ключи следует генерировать независимо друг от друга и хранить у нескольких руководителей компании, чтобы исключить возможность мошенничества со стороны одного человека. Адреса с мультиподписью также могут служить средством резервирования, когда один человек хранит несколько ключей в разных местах.

Жизнестойкость

Одним из важных аспектов безопасности, который зачастую упускается из виду, является доступность, особенно в контексте недееспособности или

смерти владельца ключа. Пользователям биткойнов рекомендуют применять сложные пароли, сохранять свои ключи в тайне и не передавать их никому. К сожалению, такая практика делает практически невозможным для семьи пользователя возврат любых средств в случае отсутствия самого пользователя для их разблокировки. Чаще всего семьи пользователей биткойнов могут вообще не подозревать о существовании таких активов.

Если у вас накопилось много биткойнов, стоит поделиться информацией о доступе к ним с доверенным родственником или адвокатом. Более сложную схему обеспечения сохранности можно организовать с помощью мультиподписи доступа и планирования наследства с помощью юриста в статусе «душеприказчика цифровых активов».

Биткойн – это сложная инновационная технология, которая все еще находится на стадии разработки. Со временем мы сможем создать более совершенные инструменты и методы повышения безопасности, более удобные для неопытных пользователей. Пока же владельцы биткойнов могут воспользоваться всеми описанными здесь рекомендациями и обеспечить себе безопасную и беспроблемную жизнь с биткойнами.

Глава 14

Приложения второго уровня

Теперь, основываясь на понимании основ системы Биткойн (первый уровень), можно взглянуть на нее как на платформу для других приложений, то есть для программ *второго уровня* (*second layer*). В этой главе Биткойн рассматривается как прикладная платформа. Разберем *примитивы* (*primitives*) для создания прикладных программ, которые формируют строительные блоки любого приложения для блокчейна. Далее будет рассмотрено несколько важных приложений, использующих эти примитивы, таких как подтверждение на стороне клиента, платежные каналы и платежные каналы с маршрутизацией (Lightning Network).

Строительные блоки (примитивы)

Система Биткойн при правильной и стабильной работе предлагает определенные гарантии, которые могут быть использованы в качестве строительных блоков для создания приложений. К ним относятся:

Отсутствие двойной оплаты (*No double spend*)
: Наиболее важная гарантия децентрализованного алгоритма консенсуса Биткойна заключается в том, что ни один UTXO не может быть потрачен дважды в одной и той же действительной цепочке блоков.

Неизменяемость (*Immutability*)
: После записи транзакции в блокчейн и добавления достаточного количества работы в последующие блоки данные транзакции становятся практически неизменяемыми. Неизменяемость обеспечивается затратами энергии, поскольку перезапись блокчейна требует значительного расхода энергии для создания доказательства работы. Необходимая энергия и, следовательно, степень неизменяемости увеличиваются с ростом объема работы, совершенной поверх блока с транзакцией.

Нейтральность (*Neutrality*)
: Децентрализованная сеть Биткойн распространяет действительные транзакции независимо от их источника. Это означает, что любой человек может

создать действительную транзакцию с достаточной комиссией и быть уверенным, что он сможет передать ее и включить в блокчейн в любое время.

Безопасная временная метка (*Secure timestamping*)
Правила консенсуса отклоняют любой блок, временная метка которого находится слишком далеко в будущем, и стремятся не пропускать блоки, временные метки которых находятся слишком далеко в прошлом. Это в определенной степени обеспечивает доверие к временным меткам блоков. Временная метка блока обозначает неизрасходованную ранее ссылку для входов всех включенных транзакций.

Авторизация (*Authorization*)
Цифровые подписи, подтвержденные в децентрализованной сети, обеспечивают гарантии авторизации. Скрипты с условием наличия цифровой подписи не могут быть выполнены без разрешения владельца секретного ключа, указанного в этом скрипте.

Контролируемость (*Auditability*)
Все транзакции являются открытыми и могут быть проверены. Все транзакции и блоки можно связать непрерывной цепочкой с блоком генезиса.

Подотчетность (*Accounting*)
В любой транзакции (кроме транзакции coinbase) стоимость входов равна стоимости выходов плюс комиссии. Невозможно создать или уничтожить стоимость биткойна в ходе транзакции. Выходы не могут быть больше входов.

Без срока действия (*Nonexpiration*)
Срок действия достоверной транзакции не ограничен. Если она действительна сегодня, то будет действительна и в ближайшем будущем, пока входы остаются неизрасходованными и правила консенсуса не меняются.

Целостность (*Integrity*)
Выходные данные транзакции Биткойна, подписанной с помощью SIGHASH_ALL, или части транзакции, подписанной с помощью другого типа SIGHASH, не могут быть изменены без признания подписи недействительной, а значит, и самой транзакции.

Неразрывность транзакций (*Transaction atomicity*)
Транзакции Биткойна атомарны. Они либо действительны и подтверждены (добыты при майнинге), либо нет. Частичные транзакции не могут майниться, и для них не существует промежуточного состояния. В любой момент времени транзакция либо майнится, либо нет.

Дискретные (*неделимые*) *единицы стоимости* (*Discrete* (*indivisible*) *units of value*)
Выходы транзакций – это дискретные и неделимые единицы стоимости. Они могут быть потрачены или не потрачены только полностью. Они не могут быть разделены или потрачены частично.

Кворум контроля (*Quorum of control*)
Ограничения мультиподписи в скриптах предусматривают кворум авторизации, предопределенный в схеме мультиподписи. Это требование обеспечивается правилами консенсуса.

Временная блокировка / срок действия (*Timelock/aging*)
Любой фрагмент скрипта с относительной или абсолютной временной блокировкой может быть выполнен только после истечения заданного срока.

Репликация (*Replication*)
Децентрализованное хранение блокчейна гарантирует, что при майнинге транзакции после достаточного количества подтверждений она реплицируется по всей сети и становится долговременной и невосприимчивой к перебоям электроснабжения, потере данных и т. д.

Защита от подделки (*Forgery protection*)
Транзакция может расходовать только уже существующие, подтвержденные выходы. Невозможно создать или подделать ее стоимость.

Согласованность (*Consistency*)
При условии отсутствия разделений майнеров возможность реорганизации или рассогласования уже записанных в блокчейн блоков уменьшается с экспоненциальной вероятностью пропорционально глубине, на которой они записаны. После достижения большой глубины записи количество вычислений и энергии, необходимых для внесения изменений, становится слишком большим для практической реализации.

Запись внешнего состояния (*Recording external state*)
Транзакция может зафиксировать значение данных через OP_RETURN или оплаты по контракту, тем самым отражая изменение состояния во внешней машине состояний.

Предсказуемая эмиссия (*Predictable issuance*)
Всего будет выпущено менее 21 млн биткойнов с предсказуемой скоростью.

Этот список строительных блоков далеко не полный, и с каждой новой функцией, появляющейся в Биткойне, добавляются новые.

Приложения из строительных блоков

Предлагаемые Биткойном строительные блоки – это элементы платформы доверия (trust platform), которые можно использовать для создания приложений. Вот несколько примеров существующих сегодня приложений и используемых в них строительных блоков:

Доказательство существования (*цифровой нотариус*)
Неизменность + временная метка + долговечность. Транзакция в блокчейне может зафиксировать определенное значение, подтверждая, что на момент записи данный фрагмент данных уже существовал (Timestamp). Обязательство не может быть изменено задним числом (Immutability), а доказательство будет храниться постоянно (Durability).

Kickstarter (*Lighthouse*)
Последовательность + атомарность + целостность. Если поставить подпись на входе и выходе (Integrity) транзакции по сбору средств, другие люди смогут

делать взносы в общий фонд, но потратить их невозможно (Atomicity) до тех пор, пока цель (сумма на выходе) не будет собрана полностью (Consistency).

Платежные каналы

Кворум контроля + блокировка по времени + запрет повторного расходования + бессрочность + устойчивость к изменениям + авторизация. Мультиподпись «2 из 2» (Quorum) с блокировкой по времени (Timelock), используемая в качестве «расчетной» транзакции платежного канала, может быть сохранена (Nonexpiration) и потрачена в любое время (Censorship Resistance) любой стороной (Authorization). Затем обе стороны могут создавать транзакции с обязательствами, которые заменяют (No Double-Spend) расчеты с более коротким временем блокировки (Timelock).

Цветные (окрашенные) монеты

Первым прикладным приложением блокчейна, которое будет рассмотрено, будут так называемые «*окрашенные*», или *цветные*, *монеты* (*colored coins*).

Цветные монеты относятся ко множеству схожих технологий, которые используют транзакции Биткойна для регистрации создания, владения и передачи сторонних активов, отличных от биткойнов. Под «сторонними» (extrinsic) здесь подразумеваются активы, которые не хранятся непосредственно в блокчейне Биткойна, в отличие от самого биткойна, который является неотъемлемым активом блокчейна.

Цветные монеты используются для отслеживания цифровых активов, а также материальных активов третьих лиц, которые продаются по сертификатам собственности на эти цветные монеты. В качестве цветных монет для цифровых активов могут выступать нематериальные активы, такие как сертификат акций, лицензия, виртуальная собственность (игровые предметы) или любая форма лицензированной интеллектуальной собственности (товарные знаки, авторские права и т. д.). Цветные монеты материальных активов могут представлять собой сертификаты собственности на товары (золото, серебро, нефть), права на землю, автомобили, лодки, самолеты и т. д.

Появление этого термина связано с идеей «окрашивания» (coloring) или маркировки номинальной суммы биткойнов, например одного сатоши, для обозначения чего-то другого, нежели сама сумма биткойнов. В качестве аналогии можно привести штамп на купюре номиналом 1 доллар с надписью «это сертификат акций ACME», или «эту купюру можно обменять на 1 унцию серебра», а затем обменять купюру номиналом 1 доллар в качестве сертификата на владение этим иным активом. Первая реализация цветных монет под названием «*улучшенное окрашивание на основе дополненного порядка*» (*Enhanced Padded-Order-Based Coloring*), или *EPOBC*, наделяла внешние активы выходом в 1 сатоши. Таким образом, это были действительно «цветные монеты», поскольку каждый актив добавлялся как атрибут (цвет) одного сатоши.

В более поздних реализациях цветных монет стали использоваться другие механизмы привязки метаданных к транзакции совместно с внешними хранилищами данных, связывающими метаданные с конкретными активами.

На сегодняшний день применяются три основных механизма: одноразовые пломбы, платеж по контракту и проверка на стороне клиента.

Одноразовые пломбы

Изначально одноразовые пломбы (single-use seals) появились как средство физической безопасности. При доставке товара через посредников возникает необходимость в способе выявления несанкционированного вскрытия. Поэтому посылка защищается специальным механизмом, который в случае вскрытия упаковки будет иметь явные повреждения. Если посылка приходит с неповрежденной пломбой, отправитель и получатель могут быть уверены, что посылка не была вскрыта во время транспортировки.

В контексте цветных монет одноразовые пломбы представляют собой структуру данных, которая может быть связана с другой структурой данных только один раз. В Биткойне этому определению соответствуют неизрасходованные транзакционные выходы (UTXO). Выходы UTXO могут быть потрачены в рамках действующего блокчейна только единожды, и процесс их траты привязывает их к данным в транзакции расходования.

На этом отчасти основан современный механизм передачи цветных монет. Одна или несколько цветных монет принимаются в UTXO. Когда этот UTXO тратится, транзакция расходования должна содержать описание способа расходования цветных монет. Это приводит нас к транзакциям «платеж по контракту» (P2C).

Платеж по контракту (P2C)

Ранее в разделе «Платеж Pay to Contract (P2C)» уже рассказывалось о транзакциях P2C, ставших частью основы для обновления правил консенсуса Биткойна с помощью метода taproot. Вкратце напомним, что P2C позволяет отправителю (Бобу) и получателю (Алисе) договориться о некоторых данных, например о контракте, а затем подстроить открытый ключ Алисы так, чтобы она приняла на себя обязательства по контракту. В любой момент Боб может раскрыть основной ключ Алисы и твик, использованный для фиксации контракта, тем самым доказав получение средств. Если Алиса расходует эти средства, это окончательно подтверждает, что она знала о контракте, поскольку единственный способ потратить средства, полученные на ключ с твиком P2C, – это знать твик (контракт).

Мощным достоинством твикнутых (настроенных) ключей P2C является то, что они выглядят как любые другие открытые ключи для всех, кроме Алисы и Боба, если только они не решат раскрыть контракт, по которому были настроены эти ключи. Ничто не будет публично раскрыто о контракте – даже сам факт существования контракта между ними.

Контракт P2C может быть произвольно длинным и подробным, его условия могут быть написаны на любом языке, и он может ссылаться на любые условия по желанию участников, поскольку контракт не проходит полную проверку на узлах, а в блокчейн отправляется только открытый ключ с обязательствами.

Если говорить о цветных монетах, Боб может открыть одноразовую пломбу, хранящую его цветные монеты с расходом соответствующего UTXO. В процессе транзакции по расходованию UTXO он может заключить контракт с указанием условий, которые должен выполнить следующий владелец (или владельцы) цветных монет, чтобы в дальнейшем их расходовать. Новым владельцем не обязательно должна быть Алиса, даже если именно Алиса получает UTXO, потраченные Бобом, и даже если именно Алиса твикнула свой открытый ключ в соответствии с условиями контракта.

Поскольку полные узлы не проверяют (и не могут проверять) соблюдение контракта, необходимо выяснить, кто несет ответственность за проверку. Это приводит нас к опции «*проверка на стороне клиента*» (*client-side validation*).

Проверка на стороне клиента

У Боба было некоторое количество цветных монет, связанных с UTXO. Он потратил эти UTXO с соблюдением обязательств по контракту, где говорится о порядке подтверждения права собственности на цветные монеты следующим получателем (или получателями) для их дальнейшего расходования.

На практике P2C-контракт Боба, скорее всего, просто обязуется предоставить один или несколько уникальных идентификаторов для UTXO, которые будут использоваться в качестве одноразовых пломб для принятия решения о следующем расходовании цветных монет. Например, в контракте Боба может быть указано, что UTXO, который Алиса получила на свой твикнутый в P2C открытый ключ, теперь контролирует половину его цветных монет, а другая половина его цветных монет теперь назначена на иной UTXO, который может не иметь никакого отношения к транзакции между Алисой и Бобом. Это обеспечивает высокую степень конфиденциальности относительно отслеживания блокчейна.

Если позже Алиса захочет отправить свои цветные монеты Дэну, ей придется сначала доказать Дэну, что она контролирует эти цветные монеты. Алиса может сделать это путем разглашения Дэну своего исходного открытого ключа P2C и условий контракта P2C, выбранных Бобом. Алиса также откроет Дэну UTXO, который Боб использовал в качестве одноразовой пломбы, и все сведения, переданные ей Бобом о предыдущих владельцах цветных монет. Словом, Алиса предоставляет Дэну полную хронологию всех предыдущих операций с цветными монетами, при этом каждый шаг фиксируется в блокчейне Биткойна (однако в цепочке не хранятся какие-либо конкретные данные – только обычные открытые ключи). Эта хронология очень похожа на историю обычных транзакций Биткойна, которую мы называем блокчейном, но история цветных монет остается совершенно невидимой для других пользователей блокчейна.

Дэн проверяет эту историю с помощью своей программы, которая носит название «*проверка на стороне клиента*» (*client-side validation*). Примечательно, что Дэну нужно получить и подтвердить только те части этой хронологии, которые имеют отношение к выбранным им цветным монетам. Ему не нужна информация о судьбе чужих цветных монет, например ему никогда не понадобится знать, что случилось со второй половиной монет Боба, которые Боб не

отдает Алисе. Это способствует повышению конфиденциальности протокола цветных монет.

Рассмотрев одноразовые пломбы, оплату по контракту и проверку на стороне клиента, можно обратиться к двум основным протоколам, использующим их в настоящее время, RGB и Taproot Assets.

Протокол RGB

Разработчики протокола RGB стали первопроходцами во многих идеях, используемых в современных протоколах цветных монет Биткойна. Главным требованием при разработке RGB была совместимость этого протокола с платежными каналами вне цепочки (см. раздел «Каналы платежей и каналы состояний» вроде используемых в Lightning Network (LN) на стр. 316). Это реализовано на каждом уровне протокола RGB:

Одноразовые пломбы
: Для создания платежного канала Боб распределяет свои цветные монеты в UTXO, для расходования которого необходимы подписи и его, и Алисы. Их взаимный контроль над этим UTXO служит одноразовой пломбой для будущих переводов.

Оплата по контракту (*P2C*)
: Теперь Алиса и Боб могут подписать несколько версий контракта P2C. Механизм принудительного исполнения, лежащий в основе платежного канала, гарантирует, что обе стороны будут заинтересованы в публикации только самой последней версии контракта на цепочке.

Проверка на стороне клиента
: Для гарантий того, что ни Алисе, ни Бобу не нужно верить друг другу на слово, каждый из них проверяет все предыдущие переводы цветных монет до момента их создания, чтобы убедиться в соблюдении всех правил контракта.

Разработчики RGB описали и другие возможности использования своего протокола, например создание токенов идентификации, которые можно периодически обновлять для защиты от компрометации секретного ключа.

Более подробную информацию можно найти в документации RGB https://rgb.tech/.

Протокол Taproot Assets

Протокол цветных монет Taproot Assets («активы главного корня»), изначально носивший название Taro, в значительной степени основан на протоколе RGB. Если сравнивать с RGB, протокол Taproot Assets использует форму контрактов P2C, которая очень похожа на версию, применяемую в taproot для реализации функциональности MAST (см. раздел «Деревья альтернативных скриптов (Merklized Alternative Script Trees, MAST)»). Заявленное преимущество Taproot Assets над RGB заключается в том, что сходство с широко используемым протоколом taproot упрощает его реализацию для кошельков и других программ.

Недостатком является меньшая гибкость, чем у протокола RGB, особенно когда речь идет о реализации неактивных функций, таких как идентификационные токены.

Протокол taproot является частью протокола Биткойна, а протокол *Taproot Assets* не является, несмотря на схожее название. И RGB, и Taproot Assets – это протоколы, разработанные поверх протокола Биткойна. Единственным активом, поддерживаемым Биткойном, являются биткойны.

Протокол Taproot Assets даже в большей степени, чем RGB, изначально разрабатывался для совместимости с LN. Одна из проблем пересылки через LN активов, не являющихся биткойнами, заключается в существовании двух способов осуществления такой пересылки, каждый из которых имеет свои компромиссы:

Нативная пересылка (*Native forwarding*)
: На каждом участке пути между отправителем и получателем необходимо иметь информацию о конкретном активе (типе цветной монеты) и наличии достаточного его баланса для пересылки платежа.

Трансляционная пересылка (*Translated forwarding*)
: Следующий за отправителем и следующий за получателем узлы должны знать о конкретных активах и иметь их достаточный баланс для пересылки платежа, однако всем остальным узлам достаточно лишь поддерживать пересылку платежей в биткойнах.

Нативная пересылка концептуально проще, но по сути требует отдельной сети типа Lightning для каждого актива. Трансляционная пересылка позволяет воспользоваться эффектом масштаба сети Биткойн LN, но может быть уязвима к проблеме так называемого «*бесплатного американского колл-опциона*» (*free American call option*), когда получатель может выборочно принимать или отклонять определенные платежи в зависимости от последних изменений обменного курса, чтобы выкачивать деньги из следующего за ним узла. Хотя идеального решения этой проблемы не существует, возможно, есть практические решения, которые позволяют ограничить ее последствия.

И Taproot Assets, и RGB технически могут поддерживать как нативную, так и трансляционную пересылку. Протокол Taproot Assets специально создан для трансляционной пересылки, в то время как в RGB есть возможность реализации обоих вариантов.

Дополнительную информацию можно найти в документации Taproot Asset https://docs.lightning.engineering/the-lightning-network/taproot-assets. Кроме того, разработчики Taproot Asset работают над выпуском протоколов BIP, которые могут появиться уже после выхода этой книги в печать.

Каналы платежей и каналы состояний

Платежные каналы (*payment channels*) – это механизм обмена биткойн-транзакциями между двумя сторонами вне блокчейна Биткойна, не требующий

доверия третьей стороны. Эти транзакции, которые были бы действительны при расчетах в блокчейне Биткойна, хранятся вне цепочки, ожидая возможного пакетного погашения. Поскольку транзакции не завершены, их можно обменивать без обычной задержки расчетов, что обеспечивает чрезвычайно высокую скорость передачи транзакций, низкие задержки и высокую степень детализации.

На самом деле термин «*канал*» в данном случае является метафорой. Каналы состояния представляют собой виртуальные конструкции для обмена статусами между двумя сторонами вне блокчейна. Как таковых «каналов» не существует, и лежащий в их основе механизм передачи данных не является таковым. Мы используем термин «*канал*» для обозначения отношений и обмена данными между двумя сторонами вне блокчейна.

Чтобы лучше объяснить эту концепцию, представьте себе поток TCP. С точки зрения протоколов более высокого уровня, это «сокет», соединяющий два приложения через интернет. Но если проследить за сетевым трафиком, то TCP-поток – это просто виртуальный канал поверх IP-пакетов. Каждая конечная точка TCP-потока упорядочивает и собирает IP-пакеты, создавая иллюзию потока байтов. В действительности же под этим скрываются отдельные пакеты. Точно так же платежный канал – это просто серия транзакций. Если они правильно упорядочены и соединены, то это создает погашаемые обязательства, которым можно доверять, даже если нет доверия к другой стороне канала.

В этом разделе будут рассмотрены различные формы каналов оплаты. Сначала будут описаны механизмы для создания одностороннего (однонаправленного) канала оплаты для дозированного сервиса микроплатежей, такого как потоковое видео. Затем этот механизм будет расширен, и будут представлены двунаправленные каналы оплаты. Наконец, речь пойдет о возможности соединения двунаправленных каналов между собой для формирования нескольких мультиузловых каналов в маршрутизируемой сети, впервые предложенной под названием *Lightning Network*.

Платежные каналы являются частью более широкой концепции *канала состояния* (*state channel*), который представляет собой изменение состояния вне цепочки, обеспеченное последующим урегулированием в блокчейне. Платежный канал – это канал состояния, в котором изменяемым состоянием является баланс виртуальной валюты.

Каналы состояния – основные принципы и терминология

Канал состояния устанавливается между двумя сторонами посредством транзакции, которая фиксирует общее состояние в блокчейне. Она называется *транзакцией финансирования* (*funding transaction*). Для создания канала эту транзакцию необходимо передать в сеть и майнить. В примере с платежным каналом заблокированное состояние – это начальный баланс (в валюте) этого канала.

Затем обе стороны обмениваются подписанными транзакциями, называемыми *транзакциями обязательств* (*commitment transactions*), которые из-

меняют начальное состояние. Эти транзакции являются действительными, поскольку они *могут* быть представлены для расчетов любой из сторон, но вместо этого они хранятся вне цепочки каждой из сторон в ожидании закрытия канала. Обновления состояния могут создаваться так быстро, насколько каждая сторона может создать, подписать и передать транзакцию другой стороне. На практике это означает, что можно обмениваться десятками транзакций в секунду.

При обмене транзакциями обязательств обе стороны также препятствуют использованию предыдущих состояний, так что наиболее актуальная транзакция обязательств всегда оказывается наиболее подходящей для погашения. Это препятствует мошенничеству любой из сторон путем одностороннего закрытия канала с предыдущим состоянием, которое для нее более выгодно, чем текущее состояние. Далее в этой главе будут рассмотрены различные механизмы препятствования публикации предыдущих состояний.

Наконец, канал может быть закрыт либо совместными усилиями, путем отправки в блокчейн последней *расчетной транзакции* (*settlement transaction*), либо в одностороннем порядке, когда любая из сторон отправляет в блокчейн последнюю транзакцию обязательств. Возможность одностороннего закрытия необходима на случай одностороннего отключения одной из сторон. Расчетная транзакция представляет собой окончательное состояние канала и выполняется в блокчейне.

За все время существования канала только две транзакции должны быть представлены для майнинга в блокчейне: транзакции финансирования и расчетов. В промежутках между этими двумя состояниями обе стороны могут обмениваться любым количеством транзакций с обязательствами, которые никто не видит и не отправляет в блокчейн.

На рис. 14.1 показан платежный канал между Бобом и Алисой, в котором отражены транзакции финансирования, принятия обязательств и расчетов.

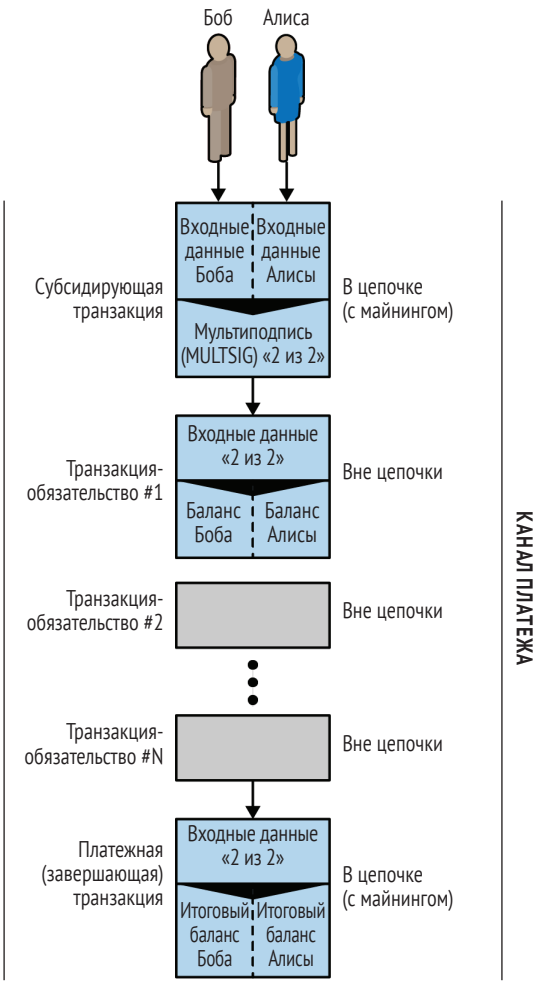


Рис. 14.1 ❖ Платежный канал между Бобом и Алисой со всеми видами транзакций

Пример простого платежного канала

Для объяснения работы каналов состояния рассмотрим очень простой пример. Это односторонний канал, то есть передача данных происходит только в одном направлении. Для простоты изложения будем исходить из простой предпосылки, что никто никого не пытается обмануть. После объяснения основной идеи канала перейдем к рассмотрению способов повышения его надежности, чтобы ни одна из сторон не могла смошенничать при всем желании.

Для этого примера возьмем двух участников: Эмму и Фабиана. Фабиан предлагает услугу потокового видео, счет за которую выставляется посекундно с использованием канала микроплатежей. Фабиан берет 0,01 миллибита (0,00001 BTC) за секунду видео, что эквивалентно 36 миллибитам (0,036 BTC) за час видео. Эмма – пользователь, который приобретает у Фабиана услугу потокового видео. На рис. 14.2 показана покупка услуги потокового видео Эммой у Фабиана с помощью канала оплаты.

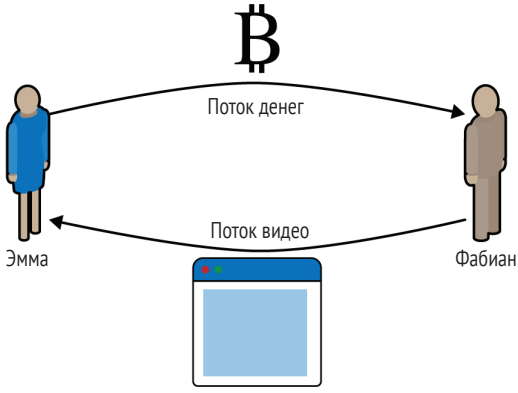


Рис. 14.2 ❖ Эмма покупает у Фабиана потоковое видео с помощью платежного канала

В этом примере Фабиан и Эмма используют специальные программы для работы с платежным каналом и потоковым видео. Эмма запускает эту программу в своем браузере, а Фабиан – на сервере. ПО включает в себя базовые функции биткойн-кошелька, может создавать и подписывать транзакции в биткойнах. Концепция и сам термин «платежный канал» полностью скрыты от пользователей. Все, что они видят, – это видео, которое оплачивается посекундно.

Для создания платежного канала Эмма и Фабиан формируют адрес с мультиподписью «2 из 2», при этом каждый из них владеет одним из ключей. В браузере Эммы отображается QR-код с адресом и запросом на внесение «депозита» за 1 час видео. После этого Эмма оплачивает адрес. Транзакция Эммы, оплачивающая адрес с мультиподписью, является финансирующей, или якорной, транзакцией (anchor transaction) для платежного канала.

Для примера предположим, что Эмма профинансировала канал 36 миллибитами (0,036 BTC). Это позволит Эмме смотреть *до* 1 часа потокового видео. Транзакция финансирования в данном случае устанавливает максимальную сумму, которая может быть передана по этому каналу, определяя тем самым емкость канала (*channel capacity*).

В процессе транзакции финансирования используется один или несколько входов из кошелька Эммы. Она создает один выход с суммой в 36 миллибит, выплачиваемой на адрес с мультиподписью «2 из 2», который контролируется совместно Эммой и Фабианом. У него могут быть дополнительные выходы для возврата средств на кошелек Эммы.

После подтверждения транзакции финансирования на достаточную глубину Эмма может приступить к просмотру потокового видео. Программа Эммы создает и подписывает транзакцию, которая изменяет баланс канала, зачисляя 0,01 миллибита на адрес Фабиана и возвращая 35,99 миллибита обратно Эмме. Подписанная Эммой транзакция потребляет выход 36 миллибит, созданный транзакцией финансирования, и создает два выхода: один для ее возврата, другой – для платежа Фабиану. Транзакция подписана лишь частично – она требует двух подписей (2 из 2), но имеет только подпись Эммы. Когда сервер Фабиана получает эту транзакцию, он добавляет вторую подпись (для входа 2 из 2) и возвращает ее Эмме вместе с видео длительностью в 1 секунду. Теперь у обеих сторон есть полностью подписанная транзакция обязательств, которую каждая из сторон может погасить самостоятельно в соответствии с актуальным балансом канала. Ни одна из сторон не транслирует эту транзакцию в сеть.

В следующем цикле ПО Эммы создает и подписывает еще одну транзакцию обязательств (обязательство № 2), которая потребляет тот же выход «2 из 2» из транзакции финансирования. Вторая транзакция-обязательство выделяет один выход в 0,02 миллибита на адрес Фабиана и один выход в 35,98 миллибита обратно на адрес Эммы. Эта новая транзакция является оплатой за две суммарные секунды видео. ПО Фабиана подписывает и возвращает вторую транзакцию обязательств вместе с еще одной секундой видео.

В таком режиме ПО Эммы продолжает отправлять транзакции с обязательствами на сервер Фабиана в обмен на потоковое видео. Баланс канала постепенно увеличивается в пользу Фабиана по мере потребления Эммой все большего количества секунд видео. Допустим, Эмма просмотрела 600 секунд (10 минут) видео, создав и подписав 600 транзакций обязательств. Последняя транзакция обязательств (#600) будет иметь два выхода, разделяя баланс канала: 6 миллибит для Фабиана и 30 миллибит для Эммы.

Наконец, Эмма нажимает «Стоп» для остановки потокового видео. Теперь Фабиан или Эмма может передать последнюю транзакцию состояния для расчета. Эта последняя транзакция является *транзакцией расчета* и оплачивает Фабиану все потребленное Эммой видео, возвращая Эмме остаток транзакции финансирования.

На рис. 14.3 показан канал между Эммой и Фабианом, а также транзакции обязательств, которые обновляют баланс канала.

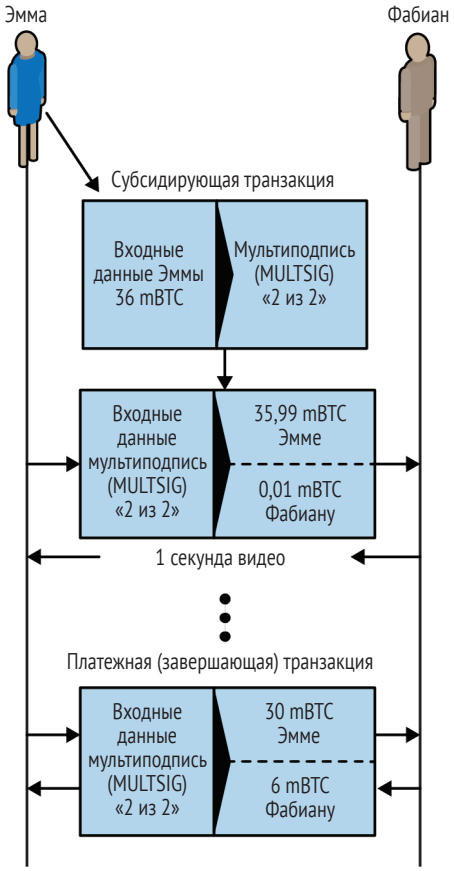


Рис. 14.3 ❖ Платежный канал Эммы и Фабиана с транзакциями обновления баланса канала

В итоге в блокчейн записываются только две транзакции: транзакция финансирования, создавшая канал, и расчетная транзакция, правильно распределившая конечный баланс между двумя участниками.

Создание бездоверительных каналов

Описанный выше канал работает только при условии сотрудничества обеих сторон, без каких-либо нарушений или попыток обмана. Рассмотрим возможные сценарии, которые нарушают работу данного канала, и посмотрим, что нужно для исправления этих ситуаций.

- Как только пройдет транзакция финансирования, Эмме понадобится подпись Фабиана для возврата части денег. Если Фабиан исчезает, средства Эммы оказываются заблокированными в схеме «2 из 2» и фактиче-

ски потеряны. В таком виде эта схема приводит к потере средств, если одна из сторон становится недоступной до подписания обеими сторонами хотя бы одной транзакции с обязательствами.

- Пока канал работает, Эмма может взять любую из транзакций с обязательствами, подписанных Фабианом, и передать одну из них в блокчейн. Зачем платить за 600 секунд видео, если она может передать транзакцию обязательства № 1 и заплатить только за 1 секунду видео? Канал не работает, потому что Эмма может обмануть, передав предварительное обязательство в свою пользу.

Обе эти проблемы можно решить с помощью таймеров. Теперь рассмотрим, как можно использовать эти таймеры на уровне транзакций.

Эмма не может пойти на риск финансирования мультиподписи «2 из 2» без гарантии возврата средств. Чтобы решить эту проблему, Эмма одновременно создает транзакцию финансирования и транзакцию возврата. Она подписывает транзакцию финансирования, но никому ее не передает. Эмма передает Фабиану только транзакцию возврата и получает его подпись.

Транзакция возврата выступает в качестве первой транзакции-обязательства, а ее временная блокировка устанавливает верхнюю границу существования канала. В данном случае Эмма может установить время блокировки на 30 дней, или 4320 блоков в будущем. Все последующие транзакции обязательств должны иметь более короткий срок блокировки по времени, чтобы они могли быть погашены до транзакции возврата.

Теперь, когда у Эммы есть полностью подписанная транзакция возврата, она может смело передавать подписанную транзакцию финансирования, будучи уверенной, что в конечном итоге, после истечения срока блокировки, она сможет погасить транзакцию возврата, даже если Фабиан исчезнет.

Каждая транзакция обязательств, которыми стороны обмениваются в период существования канала, будет заблокирована по времени в будущем. Но задержка будет немного короче для каждого обязательства, поэтому самое последнее обязательство может быть погашено раньше, чем предыдущее, которое оно аннулирует. Из-за времени блокировки ни одна из сторон не сможет успешно распространить ни одну из транзакций с обязательствами, пока не истечет время их блокировки. Если все пойдет успешно, они смогут наладить сотрудничество и спокойно закрыть канал с помощью расчетной транзакции, что избавит их от необходимости передавать промежуточную транзакцию обязательств. В противном случае можно передать самую последнюю транзакцию с обязательствами для урегулирования счета и аннулирования всех предыдущих транзакций с обязательствами.

Например, если транзакция обязательства № 1 зафиксирована по времени на 4320 блоков в будущем, то транзакция обязательства № 2 зафиксирована по времени на 4319 блоков в будущем. Транзакция принятия обязательств № 600 может быть проведена за 600 блоков до того, как транзакция принятия обязательств № 1 станет действительной.

На рис. 14.4 показано, что каждая транзакция обязательств устанавливает более короткую временную блокировку, что позволяет потратить ее до того, как предыдущие обязательства станут действительными.

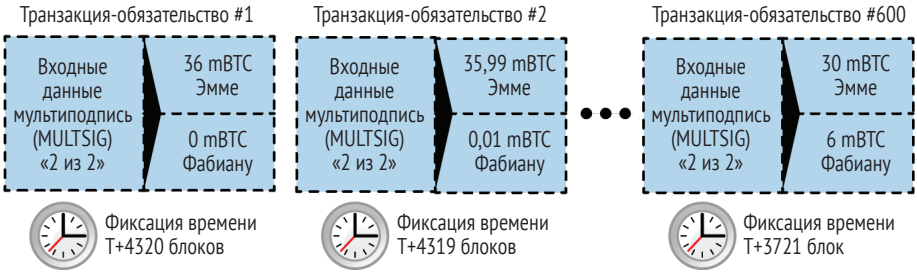


Рис. 14.4 ❖ Каждое следующее обязательство устанавливает более короткий срок блокировки

Каждая последующая транзакция обязательства должна иметь более короткий срок блокировки, чтобы она могла быть передана раньше своих предшественников и раньше транзакции возврата. Более ранняя передача обязательства гарантирует возможность расходования финансовых средств и исключает вероятность погашения любой другой транзакции обязательства за счет расходования выхода. Благодаря гарантиям блокчейна Биткойна по предотвращению двойных трат и соблюдению временных блокировок каждая транзакция обязательства фактически аннулирует своих предшественников.

Каналы состояний используют временные блокировки для реализации смарт-контрактов по времени. В этом примере показано, как временной аспект обеспечивает действительность самой последней транзакции обязательства раньше всех предыдущих. Таким образом, самая последняя транзакция обязательств может передаваться с расходованием средств и аннулированием предыдущих транзакций обязательств. Исполнение смарт-контрактов с абсолютными временными блокировками защищает от обмана одной из сторон. В этой реализации не требуется ничего, кроме абсолютного времени блокировки на уровне транзакций. Далее будет показано, как временные блокировки на уровне скриптов, `CHECKLOCKTIMEVERIFY` и `HECKSEQUENCEVERIFY`, можно использовать для построения более гибких, удобных и сложных каналов состояния.

Временные блокировки не являются единственным способом отмены предыдущих транзакций с обязательствами. В следующих разделах будет показано, как для достижения аналогичного результата можно использовать ключ отзыва (revocation key). Временные блокировки весьма эффективны, но у них есть два существенных недостатка. Устанавливая максимальную временную блокировку при первом открытии канала, они тем самым ограничивают время его существования. Что еще хуже, они требуют от реализаций каналов соблюдать баланс между длительным сроком существования каналов и длительным ожиданием возврата средств одним из участников в случае преждевременного завершения. Например, если позволить каналу оставаться открытым в течение 30 дней, установив временную блокировку возврата средств на 30 дней,

то в случае внезапного исчезновения одной из сторон другой придется ждать возврата средств все эти 30 дней. Чем дальше конечная точка, тем более отдален срок возврата.

Вторая проблема заключается в потере времени, поскольку каждая последующая транзакция обязательств должна уменьшать время блокировки, возникает явное ограничение на количество транзакций обязательств, которыми стороны успеют обменяться. Например, 30-дневный канал с временной блокировкой на 4320 блоков в будущем может принять только 4320 промежуточных транзакций обязательств до своего закрытия. Установка интервала между транзакциями обязательства по временной блокировке в 1 блок таит в себе определенную опасность. Устанавливая интервал между транзакциями обязательства в 1 блок, разработчик создает очень большую нагрузку на участников канала, которым придется быть бдительными, оставаться онлайн и следить за происходящим, а также быть готовыми в любой момент передать нужную транзакцию обязательства.

В предыдущем примере с однонаправленным каналом временную блокировку на выполнение обязательств устранить несложно. После получения Эммой от Фабиана подписи под транзакцией возврата с временной блокировкой на транзакции с обязательствами временные блокировки не устанавливаются. Напротив, Эмма отправляет Фабиану свою подпись под каждой транзакцией с обязательствами, но Фабиан не отправляет ей ни одной из своих подписей под транзакциями с обязательствами. Это означает, что только Фабиан имеет обе подписи под транзакциями обязательств, поэтому только он может передать одно из этих обязательств. Когда Эмма закончит просмотр потокового видео, Фабиан всегда предпочтет передать транзакцию, которая заплатит ему больше всего, то есть самое последнее состояние. Такая конструкция называется платежным каналом Спиллмана (Spillman-style payment channel), который впервые был описан и реализован в 2013 году, хотя безопасно использовать его можно только с транзакциями-свидетелями (segwit), которые стали доступны лишь в 2017 году.

Теперь, когда понятен принцип использования временных блокировок для отмены предыдущих обязательств, можно оценить разницу между совместным закрытием канала и закрытием канала в одностороннем порядке путем передачи транзакции обязательства. Все транзакции обязательств в предыдущем примере были заблокированы по времени, поэтому передача транзакции обязательств всегда будет связана с ожиданием истечения времени блокировки. Но если две стороны согласны с окончательным балансом и знают об имеющихся у них транзакциях обязательств, которые в конечном итоге сделают этот баланс реальностью, они могут создать расчетную транзакцию без временной блокировки с тем же балансом. При совместном закрытии любая из сторон берет последнюю транзакцию обязательства и создает расчетную транзакцию, которая идентична во всех отношениях, за исключением отказа от блокировки по времени. Обе стороны могут подписать эту расчетную транзакцию в уверенности, что не существует способа обмануть и получить более выгодный баланс. Совместно подписав и передав расчетную транзакцию, они могут закрыть канал и немедленно пополнить свой баланс. В худшем случае одна из

сторон может проявить мелочность, отказаться от сотрудничества и заставить другую сторону выполнить одностороннее закрытие с помощью самой последней транзакции обязательства. В этом случае им тоже придется дожидаться своих средств.

Асимметричные отзывные обязательства

Другим способом работы с предыдущими состояниями обязательств является их явная отмена. Однако добиться этого не так-то просто. Ключевой характеристикой Биткойна является положение о том, что если транзакция считается действительной, то она остается действительной бессрочно. Единственный способ отменить транзакцию – получить подтверждение противоречащей ей транзакции. Именно поэтому в примере с простым платежным каналом использовались временные блокировки. Они гарантировали, что недавние обязательства могут быть потрачены до того, как старые обязательства станут действительными. Однако последовательность обязательств во времени создает ряд ограничений, затрудняющих использование платежных каналов.

Несмотря на невозможность отмены транзакции, ее можно создать таким образом, чтобы ее использование стало нежелательным. Для этого каждой стороне дается *ключ отзыва* (*revocation key*), который можно использовать для пресечения действий другой стороны, если она попытается обмануть. Этот механизм отзыва предыдущих транзакций с обязательствами был впервые предложен в рамках LN.

Для объяснения сути ключей отзыва рассмотрим более сложный платежный канал между двумя биткойн-биржами, которыми Хитеш и Айрин управляют в Индии и США соответственно. Клиенты индийской биржи Хитеша часто отправляют платежи клиентам американской биржи Айрин, и наоборот. В настоящее время эти транзакции осуществляются на блокчейне Биткойна, но это подразумевает оплату комиссии и ожидание подтверждений в несколько блоков. Создание платежного канала между биржами существенно снизит затраты и ускорит поток транзакций.

Хитеш и Айрин открывают канал совместной транзакцией финансирования, каждый из них предоставляет каналу по 5 биткойнов. Прежде чем подписать транзакцию финансирования, они должны подписать первый набор обязательств (называемый *возвратом* – *refund*), в котором указывается начальный баланс в 5 биткойнов для Хитеша и 5 биткойнов для Айрин. Транзакция финансирования фиксирует состояние канала с мультиподписью «2 из 2», как и в примере с простым каналом.

Транзакция финансирования может иметь один или несколько входов от Хитеша (добавление до 5 биткойнов или более) и один или несколько входов от Айрин (добавление до 5 биткойнов или более). Количество входов должно немного превышать емкость канала для покрытия комиссии за транзакции. Транзакция имеет один выход, который фиксирует 10 биткойнов на адресе с мультиподписью «2 из 2», контролируемом как Хитешем, так и Айрин. Транзакция финансирования также может иметь один или несколько выходов, возвращающих Хитешу и Айрин изменения, если их входы превысили намеченный размер вклада в канал. Это одна транзакция с входом, предложенным

и подписанным двумя сторонами. Она должна быть создана совместно и подписана каждой стороной до передачи.

Теперь вместо создания единой транзакции обязательств, которую подписывают обе стороны, Хитеш и Айрин создают две разные транзакции обязательств, которые являются *асимметричными* (*asymmetric*).

У Хитеша есть транзакция обязательств с двумя выходами. Первый выход выплачивает Айрин причитающиеся ей 5 биткойнов *немедленно*. Второй выход выплачивает Хитешу причитающиеся ему 5 биткойнов, но только после временной блокировки в 1000 блоков. Выходные данные транзакции выглядят следующим образом:

```
Input: 2-of-2 funding output, signed by Irene

Output 0 <5 bitcoins>:
    <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoins>:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key> CHECKSIG
```

У Айрин есть другая транзакция с обязательствами и двумя выходами. Первый выход выплачивает Хитешу причитающиеся ему 5 биткойнов немедленно. Второй выход выплачивает Айрин причитающиеся ей 5 биткойнов, но только после временной блокировки в 1000 блоков. Транзакция с обязательствами, которую проводит Айрин (с подписью Хитеша), выглядит следующим образом:

```
Input: 2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoins>:
    <Hitesh's Public Key> CHECKSIG

Output 1 <5 bitcoins>:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Irene's Public Key> CHECKSIG
```

Таким образом, каждая сторона проводит транзакцию с обязательствами, расходуя выход финансирования «2 из 2». Этот вход подписывается *другой* стороной. В любой момент времени владелец транзакции может также подписать ее (завершив «2 из 2») и передать ее. Однако если они передают транзакцию с обязательством, она оплачивается другой стороной немедленно, в то время как им приходится ждать истечения времени блокировки. Установив задержку на погашение одного из выходов, можно поставить каждую сторону в невыгодное положение, если она решит передать транзакцию с обязательствами в одностороннем порядке. Однако для поощрения честного поведения одной лишь временной задержки недостаточно.

На рис. 14.5 показаны две асимметричные транзакции с обязательствами с задержкой выхода, выплачиваемого держателю обязательства.

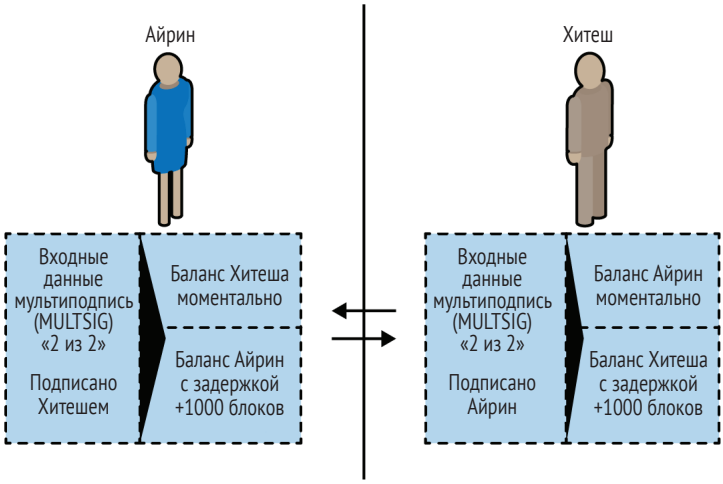


Рис. 14.5 ❖ Две транзакции с асимметричными обязательствами и отложенным платежом

Теперь вводим последний элемент этой схемы: ключ отзыва, который не позволяет мошеннику передать просроченное обязательство. Ключ отзыва позволяет пострадавшей стороне оштрафовать мошенника, забрав весь баланс канала.

Ключ отзыва состоит из двух секретов, каждая половина которых генерируется независимо всеми участниками канала. Он похож на мультиподпись «2 из 2», но построен с использованием арифметики на эллиптических кривых, так что обе стороны знают открытый ключ отзыва, но каждая сторона знает только половину секретного ключа отзыва.

В каждом цикле обе стороны раскрывают свою половину секретного ключа отзыва другой стороне, тем самым давая другой стороне (у которой теперь есть обе половины) возможность потребовать штрафной выход, если эта транзакция отзыва когда-либо будет передана.

Каждая из транзакций обязательств имеет «отложенный» выход. Скрипт выхода позволяет одной стороне выкупить его через 1000 блоков или другой стороне выкупить его при наличии ключа отзыва, что влечет за собой штраф за передачу отозванного обязательства.

Поэтому когда Хитеш создает транзакцию с обязательствами для Айрин, которую она должна подписать, он делает второй выход оплачиваемым для себя через 1000 блоков или для открытого ключа отзыва (ему известна только половина секрета). Хитеш создает эту транзакцию. Он раскроет свою половину секрета отзыва Айрин только в том случае, если будет готов перейти к новому состоянию канала и захочет отозвать это обязательство.

Скрипт второго выхода выглядит следующим образом:

```
Output 0 <5 bitcoins>:
    <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoins>:
IF
```

```
    # Revocation penalty output
    <Revocation Public Key>
ELSE
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key>
ENDIF
CHECKSIG
```

Айрин может спокойно подписать эту транзакцию, поскольку в случае ее передачи она немедленно выплатит ей причитающиеся деньги. Хитеш удерживает транзакцию, но знает, что в случае ее передачи при одностороннем закрытии канала ему придется ждать 1000 блоков для получения оплаты.

После перехода канала в следующее состояние Хитеш должен отозвать эту транзакцию-обязательство, прежде чем Айрин согласится подписать любые другие транзакции-обязательства. Для этого ему достаточно отправить свою половину ключа отзыва Айрин. Как только она получит обе половины секретного ключа отзыва этого обязательства, она сможет спокойно подписывать будущие обязательства. Она знает, что если Хитеш попробует обмануть, опубликовав предыдущее обязательство, она сможет использовать ключ отзыва для погашения отложенного вывода Хитеша. *Если Хитеш схитрит, Айрин получит ОБА выхода*. В то же время у Хитеша есть только половина секрета отзыва для этого открытого ключа отзыва, и он не сможет погасить выход до истечения 1000 блоков. Айрин сможет выкупить выход и оштрафовать Хитеша до истечения 1000 блоков.

Протокол отзыва является двусторонним. Это означает, что в каждом цикле, по мере изменения состояния канала, обе стороны обмениваются новыми обязательствами, секретами отзыва предыдущих обязательств и подписывают новые транзакции друг друга. После перехода в новое состояние они делают невозможным использование предыдущего состояния, передавая друг другу необходимые секреты отзыва для пресечения любого мошенничества.

Рассмотрим это на примере. Один из клиентов Айрин хочет отправить 2 биткойна одному из клиентов Хитеша. Для передачи 2 биткойнов по каналу Хитеш и Айрин должны изменить состояние канала, чтобы отразить новый баланс. Они перейдут в новое состояние (состояние номер 2), в котором 10 биткойнов канала будут разделены: 7 биткойнов достанутся Хитешу и 3 биткойна перейдут к Айрин. Для перехода в новое состояние канала каждый из них создаст новые транзакции-обязательства, отражающие новый баланс канала.

Как и прежде, эти транзакции с обязательствами асимметричны, поэтому транзакции с обязательствами у каждой стороны вынуждают их ожидать в случае их погашения. Очень важно, что перед подписанием новых транзакций с обязательствами они должны сначала обменяться ключами отзыва для признания недействительными всех устаревших обязательств. В данном конкретном случае интересы Хитеша совпадают с действительным состоянием канала, поэтому у него нет причин передавать предыдущее состояние. Однако для Айрин состояние 1 оставляет ей более высокий баланс, чем состояние 2. Когда Айрин передает Хитешу ключ отзыва для своей предыдущей транзакции (состояние 1), она фактически лишает себя возможности получать прибыль от

возвращения канала в предыдущее состояние, поскольку с ключом отзыва Хитеш может без задержки погасить оба выхода предыдущей транзакции. То есть если Айрин передаст предыдущее состояние, Хитеш сможет воспользоваться своим правом забрать все выходы.

Важно отметить, что отзыв не происходит автоматически. Хотя у Хитеша есть возможность оштрафовать Айрин за обман, он должен внимательно следить за блокчейном для выявления признаков обмана. Если он увидит передачу транзакции с предыдущим обязательством, у него будет в запасе 1000 блоков, чтобы принять меры и использовать ключ отзыва против попытки мошенничества Айрин, и оштрафовать ее, забрав весь баланс, все 10 биткойнов. Асимметричные отзывные обязательства (Asymmetric revocable commitments) с относительными временными блокировками (CSV) – это гораздо лучший способ реализации платежных каналов и очень значительная инновация в этой сфере. При такой конструкции канал может оставаться открытым неограниченно долго и иметь миллиарды промежуточных транзакций с обязательствами. В реализациях LN состояние обязательства идентифицируется 48-битным индексом, что позволяет осуществлять более 281 трлн ($2{,}8 \times 10^{14}$) переходов состояния в любом отдельном канале.

Контракты с хеш-таймером (HTLC)

Платежные каналы могут быть расширены с помощью специального типа смарт-контракта, который позволяет участникам выделять средства под выкупаемый секрет с указанием срока действия. Эта функция называется «*контракт с хеш-блокировкой по времени*» (*hash time lock contract*, *HTLC*) и используется как в двунаправленных, так и в маршрутизируемых платежных каналах.

Сначала разберемся, что такое «хеш» в HTLC. Чтобы создать HTLC, предполагаемый получатель платежа сначала создает секрет *R*. Затем он вычисляет хеш этого секрета, *H*:

$$H = Hash(R).$$

В результате получается хеш *H*, который можно включить в скрипт выхода. Знающий секрет может использовать его для погашения выхода. Секрет *R* также называют предварительным образом (преобраз, preimage) хеш-функции. Преобраз – это просто данные, которые используются в качестве входа для хеш-функции.

Вторая часть HTLC – это компонент «блокировки по времени». Если секрет не раскрыт, плательщик HTLC через некоторое время может получить «возврат». Это достигается с помощью абсолютной временной блокировки с использованием `CHECKLOCKTIMEVERIFY`.

Скрипт, реализующий HTLC, может выглядеть следующим образом:

```
IF
    # Payment if you have the secret R
    HASH160 <H> EQUALVERIFY
    <Receiver Public Key> CHECKSIG
ELSE
    # Refund after timeout.
```

```
    <lock time> CHECKLOCKTIMEVERIFY DROP
    <Payer Public Key> CHECKSIG
ENDIF
```

Знающий секрет *R*, который при хешировании равен *H*, может выкупить этот выход, воспользовавшись первым пунктом потока `IF`.

Если секрет не раскрыт и HTLC заявлен после определенного количества блоков, плательщик может потребовать возврат средств, используя второй пункт потока `IF`.

Это базовая реализация HTLC. Такой тип HTLC может быть выкуплен любым, кто владеет секретом *R*. HTLC может принимать множество различных форм с небольшими изменениями в скрипте. Например, добавление оператора `CHECKSIG` и открытого ключа в первом пункте ограничивает выкуп хеша определенным получателем, который также должен знать секрет *R*.

Маршрутизированные платежные каналы (Lightning Network)

Lightning Network (LN) – это предложенная маршрутизируемая сеть двунаправленных платежных связанных каналов. Такая сеть позволяет любому участнику направлять платеж из канала в канал без необходимости доверять посредникам. Впервые LN была описана https://lightning.network/lightning-network-paper.pdf Джозефом Пуном (Joseph Poon) и Тадеусом Драйей (Thadeus Dryja) в феврале 2015 года исходя из концепции платежных каналов, предложенной и доработанной многими другими.

Под термином «Lightning Network» подразумевается определенная структура сети маршрутизируемых платежных каналов, которая на данный момент реализована как минимум пятью различными командами в открытом исходном коде. Независимые реализации координируются набором стандартов совместимости, которые описаны в *репозитории Basics of Lightning Technology* (*BOLT*) https://github.com/lightning/bolts/blob/master/00-introduction.md.

Базовый пример Lightning Network

Давайте посмотрим, как это работает.

В этом примере есть пять участников: Алиса, Боб, Кэрол, Диана и Эрик. Эти пять участников попарно открыли друг у друга платежные каналы. У Алисы есть платежный канал с Бобом. Боб подключен к Кэрол, Кэрол к Диане, а Диана к Эрику. Для простоты предположим, что каждый канал пополняется каждым участником на 2 биткойна, что в общей сложности составляет 4 биткойна в каждом канале.

На рис. 14.6 показаны пять участников LN, соединенных двунаправленными платежными каналами. Они могут быть связаны для осуществления платежа от Алисы к Эрику (см. «Каналы маршрутизируемых платежей (Lightning Network)»).

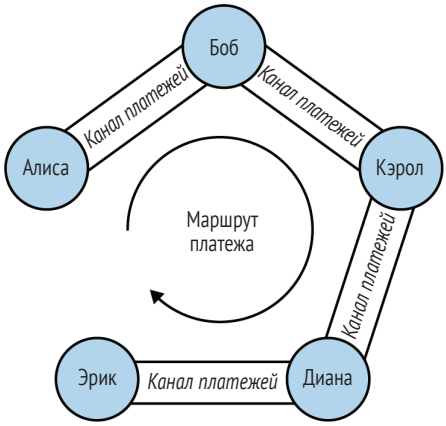


Рис. 14.6 ❖ Серия двунаправленных платежных каналов, соединенных в LN

Алиса хочет заплатить Эрику 1 биткойн. Однако Алиса не связана с Эриком платежным каналом. Создание платежного канала требует транзакции финансирования, которая должна быть зафиксирована в блокчейне Биткойна. Алиса не хочет открывать новый платежный канал и тратить больше своих средств. Есть ли способ заплатить Эрику косвенным образом?

На рис. 14.7 показан пошаговый процесс маршрутизации платежа от Алисы к Эрику через серию HTLC-обязательств на соединяющих участников платежных каналах.

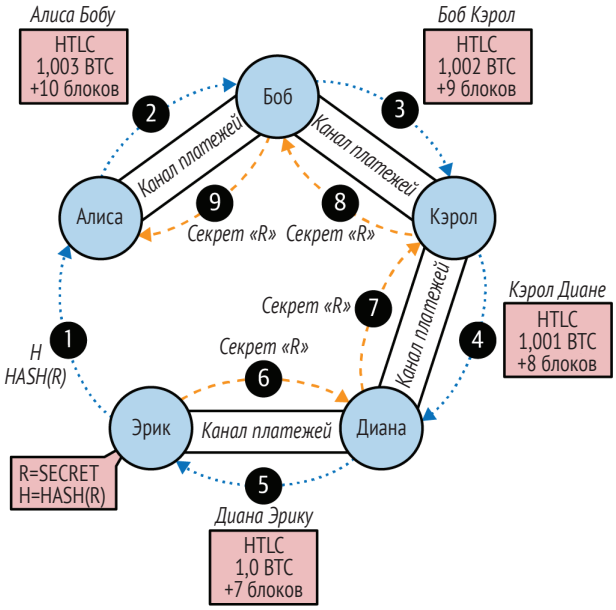


Рис. 14.7 ❖ Пошаговая маршрутизация платежа через LN

Алиса управляет узлом LN, который отслеживает ее платежный канал для Боба и имеет возможность обнаруживать маршруты между платежными каналами. LN-узел Алисы также имеет возможность подключаться через интернет к LN-узлу Эрика. LN-узел Эрика создает секрет R с помощью генератора случайных чисел. Узел Эрика не раскрывает этот секрет никому. Вместо этого узел Эрика вычисляет хеш H секрета R и передает этот хеш узлу Алисы в виде счета (см. рис. 14.7, шаг 1).

Теперь LN-узел Алисы строит маршрут между LN-узлом Алисы и LN-узлом Эрика. Используемый алгоритм поиска пути будет рассмотрен более подробно позже, а пока предположим, что узел Алисы способен найти эффективный маршрут.

Затем узел Алисы строит HTLC, оплачиваемый хешем H, с тайм-аутом возврата в 10 блоков (текущий блок + 10), на сумму 1,003 биткойна (см. рис. 14.7, шаг 2). Дополнительные 0,003 биткойна пойдут на компенсацию промежуточным узлам за их участие в этом платежном маршруте. Алиса предлагает этот HTLC Бобу, вычитая 1,003 биткойна из баланса своего канала с Бобом и передавая их в HTLC. HTLC имеет следующее значение: «*Алиса передает 1,003 биткойна из баланса своего канала, которые будут выплачены Бобу, если Боб знает секрет, или возвращены на баланс Алисы, если пройдет 10 блоков*». Теперь баланс канала между Алисой и Бобом выражается транзакциями обязательств с тремя выходами: 2 биткойна на балансе Боба, 0,997 биткойна на балансе Алисы, 1,003 биткойна на HTLC Алисы. Баланс Алисы уменьшается на зафиксированную в HTLC сумму.

Теперь у Боба есть обязательство, согласно которому, если он сможет получить секрет R в течение следующих 10 блоков, он сможет получить 1,003 биткойна, заблокированных Алисой. Имея на руках это обязательство, узел Боба строит HTLC на своем платежном канале с Кэрол. HTLC Боба обязуется передать 1,002 биткойна в хеш H на 9 блоков, которые Кэрол сможет выкупить, если у нее есть секрет R (см. рис. 14.7, шаг 3). Боб знает, что если Кэрол может претендовать на его HTLC, она должна выдать R. Если у Боба есть R в девяти блоках, он может использовать его, чтобы претендовать на HTLC Алисы. Он также зарабатывает 0,001 биткойна за использование баланса своего канала в течение девяти блоков. Если Кэрол не сможет потребовать свой HTLC, а он не сможет потребовать HTLC Алисы, все возвращается к прежним балансам каналов, и никто не остается в убытке. Баланс канала между Бобом и Кэрол теперь таков: 2 у Кэрол, 0,998 у Боба, 1,002 у Боба на HTLC.

Теперь у Кэрол есть обязательство, по которому, если она получит R в течение следующих девяти блоков, она сможет претендовать на 1,002 биткойна, заблокированных Бобом. И она может взять на себя обязательство HTLC на своем канале с Дианой. Она берет на себя обязательство HTLC в размере 1,001 биткойна на хеш H, на восемь блоков, который Диана может выкупить, если у нее есть секрет R (см. рис. 14.7, шаг 4). С точки зрения Кэрол, если это сработает, она станет богаче на 0,001 биткойна, а если нет, то ничего не потеряет. Ее HTLC для Дианы станет возможным только в том случае, если R будет раскрыт, и тогда она сможет потребовать HTLC у Боба. Баланс каналов между Кэрол и Дианой теперь таков: 2 у Дианы, 0,999 у Кэрол, 1,001 у Кэрол на HTLC.

Наконец, Диана может предложить HTLC Эрику, выделив 1 биткойн на семь блоков для хеша H (см. рис. 14.7, шаг 5). Баланс каналов между Дианой и Эриком теперь составляет: 2 у Эрика, 1 у Дианы, 1 у Дианы на HTLC.

Однако на этом участке маршрута у Эрика *есть* секрет R. Поэтому он может претендовать на HTLC, предложенный Дианой. Он отправляет R Диане и запрашивает 1 биткойн, добавляя его к балансу своего канала (см. рис. 14.7, шаг 6). Баланс канала сейчас составляет: 1 у Дианы, 3 у Эрика.

Теперь у Дианы есть секрет R. Поэтому она может требовать HTLC от Кэрол. Диана передает R Кэрол и добавляет 1,001 биткойна к балансу своего канала (см. рис. 14.7, шаг 7). Теперь баланс канала между Кэрол и Дианой составляет: 0,999 у Кэрол, 3,001 у Дианы. Диана «заработала» 0,001 за участие в этом маршруте платежей.

Проходя обратно по маршруту, секрет R позволяет каждому участнику потребовать оставшиеся HTLC. Кэрол требует 1,002 от Боба, в результате чего баланс на их канале составляет: 0,998 у Боба, 3,002 у Кэрол (см. рис. 14.7, шаг 8). Наконец, Боб требует HTLC от Алисы (см. рис. 14.7, шаг 9). Баланс их каналов обновляется следующим образом: 0,997 у Алисы, 3,003 у Боба.

Алиса заплатила Эрику 1 биткойн без открытия канала для Эрика. Ни одна из промежуточных сторон в маршруте платежа не должна была доверять друг другу. За краткосрочное размещение своих средств в канале они могут получить небольшую комиссию, при этом единственным риском является небольшая задержка возврата средств в случае закрытия канала или неудачи маршрутизированного платежа.

Транспорт и маршрутизация в Lightning Network

Все коммуникации между узлами LN шифруются по принципу «точка-точка». Кроме того, узлы имеют долгосрочный открытый ключ, который они используют в качестве идентификатора и для аутентификации друг друга.

Когда узел намеревается отправить платеж другому узлу, он должен сначала построить *путь* через сеть, соединив платежные каналы с достаточной емкостью. Узлы публикуют информацию о маршрутизации, в том числе о том, какие каналы у них открыты, какова их емкость, а также о размере комиссии за маршрутизацию платежей. Информация о маршрутизации может передаваться несколькими способами, и по мере развития технологии LN появились различные протоколы поиска маршрута. Современные реализации поиска маршрутов используют модель P2P, в которой узлы распространяют объявления о каналах среди своих пиров по принципу «флуда», подобно тому, как Биткойн распространяет транзакции.

В предыдущем примере узел Алисы использует один из этих механизмов обнаружения маршрутов для определения одного или нескольких путей, соединяющих ее узел с узлом Эрика. Как только узел Алисы построит маршрут, он инициализирует его в сети, распространяя серию зашифрованных и вложенных инструкций для подключения каждого из смежных платежных каналов.

Важно отметить, что этот путь известен только узлу Алисы. Все остальные участники маршрута платежа видят только смежные узлы. С точки зрения Кэ-

рол, это выглядит как платеж от Боба к Диане. Кэрол не знает, что Боб на самом деле передает платеж от Алисы. Она также не знает, что Диана будет передавать платеж Эрику.

Это критически важная особенность LN, поскольку она обеспечивает конфиденциальность платежей и затрудняет применение слежки, цензуры или черных списков. Но как Алисе установить этот путь платежа, ничего не раскрывая узлам-посредникам?

В LN реализован протокол «луковичной маршрутизации» (onion-routed protocol) на базе схемы под названием Sphinx. Этот протокол маршрутизации гарантирует, что отправитель платежа может построить и передать через LN такой путь, что:

- промежуточные узлы могут проверять и расшифровывать свою часть информации о маршруте, определяя следующий узел;
- кроме предыдущего и следующего узлов, они не могут узнать что-либо о других узлах на маршруте;
- они не могут определить длину платежного пути или свое собственное положение на этом пути;
- каждая часть пути зашифрована таким образом, чтобы злоумышленник на уровне сети не смог связать пакеты из разных частей пути друг с другом;
- в отличие от Tor (протокол анонимизации интернета с «луковой маршрутизацией»), здесь нет «выходных узлов», за которыми можно было бы установить наблюдение. Платежи не нужно передавать в блокчейн Биткойна; узлы просто обновляют балансы каналов.

Используя этот протокол с луковичной маршрутизацией, Алиса «оборачивает» каждый элемент пути слоем шифрования, начиная с конца и двигаясь в обратном направлении. Она шифрует сообщение Эрику с помощью открытого ключа Эрика. Это сообщение обертывается в зашифрованное сообщение Диане, идентифицируя Эрика как следующего получателя. Сообщение Диане заворачивается в сообщение, зашифрованное открытым ключом Кэрол и идентифицирующее Диану как следующего получателя. Сообщение Кэрол зашифровано ключом Боба. Таким образом, Алиса создала эту зашифрованную многослойную «луковицу» из сообщений. Она отправляет ее Бобу, который может расшифровать и развернуть только внешний слой. Внутри Боб находит сообщение Кэрол, которое он может ей переслать, но не может расшифровать сам. Следуя по этому пути, сообщения пересылаются, расшифровываются, пересылаются далее и т. д., вплоть до Эрика. Каждый участник знает только предыдущий и следующий узлы в каждом переходе.

Каждый элемент маршрута содержит информацию о HTLC, который должен быть продлен до следующего узла, об отправляемой сумме, о включаемой комиссии и об истечении времени блокировки CLTV (в блоках) HTLC. По мере распространения информации о маршруте узлы берут на себя обязательства по передаче HTLC до следующего узла.

На этом этапе может возникнуть недоумение, как это возможно, что узлы не знают длину пути и свое положение на этом пути. В конце концов, они по-

лучают сообщение и передают его следующему узлу. Разве оно не становится короче, что позволяет им определить размер пути и свое положение? Чтобы предотвратить это, размер пакета фиксирован и заполнен случайными данными. Каждый узел видит следующий переход и зашифрованное сообщение фиксированной длины для пересылки. Только конечный получатель видит, что следующего перехода нет. Всем остальным кажется, что впереди еще много переходов.

Преимущества Lightning Network

LN – это технология маршрутизации второго уровня. Она может использоваться с любым блокчейном, поддерживающим некоторые базовые возможности, такие как транзакции с мультиподписью, блокировка по времени и базовые смарт-контракты.

LN располагается поверх сети Биткойна, что позволяет значительно увеличить пропускную способность, конфиденциальность, детализацию и скорость, не жертвуя при этом принципами работы без участия посредников:

Конфиденциальность (*Privacy*)
: Платежи LN намного более конфиденциальны, чем платежи в блокчейне Биткойна, поскольку они не являются открытыми. Хотя участники маршрута могут видеть платежи, распространяемые по их каналам, они не знают ни отправителя, ни получателя.

Удобство использования (*Fungibility*)
: LN значительно затрудняет отслеживание и создание «черных списков» в Биткойне, повышая удобство использования этой валюты.

Скорость (*Speed*)
: Транзакции Биткойна с использованием LN осуществляются за миллисекунды, а не за минуты или часы, поскольку HTLC проводятся без фиксации транзакций в блокчейне.

Гранулярность (*Granularity*)
: LN позволяет осуществлять платежи как минимум такого же размера, как лимит «пыли» Биткойна, а возможно, и меньше.

Производительность (*Capacity*)
: LN увеличивает производительность системы Биткойн на несколько порядков. Верхняя граница количества платежей в секунду, которые могут быть проведены через Lightning Network, зависит только от мощности и скорости каждого узла.

Работа без доверия (*Trustless Operation*)
: LN использует транзакции Биткойна между узлами, которые работают как равные, не доверяя друг другу. Таким образом, LN сохраняет принципы системы Биткойн, но при этом значительно расширяет ее рабочие параметры.

Здесь были рассмотрены лишь некоторые из новых приложений, которые могут быть созданы с использованием блокчейна Биткойна в качестве плат-

формы доверия. Эти приложения расширяют сферу применения Биткойна за пределы сферы платежных операций.

Что вы будете делать с полученными знаниями теперь, когда дочитали эту книгу до конца? Миллионы людей, возможно миллиарды, знают слово «Биткойн», но лишь небольшой процент из них знает о принципах работы Биткойна столько же, сколько вы сейчас. Эти знания очень ценны. Еще более ценными являются люди, такие как вы, которые настолько увлечены Биткойном, что готовы прочитать о нем несколько сотен страниц.

Если вы еще не сделали этого, пожалуйста, подумайте о возможности внести какой-либо вклад в развитие Биткойна. Вы можете запустить полноценный узел для подтверждения получаемых вами платежей в биткойнах, создать приложения, которые облегчат другим работу с Биткойном, или помочь в ознакомлении других людей с Биткойном и его потенциалом. Вы даже можете сделать редкий шаг – внести свой вклад в программное обеспечение инфраструктуры Биткойна с открытым исходным кодом, такое как Bitcoin Core, активно сотрудничая с небольшим количеством невероятно умных людей, чтобы создать инструменты, за которые никто никогда не заплатит, но от которых однажды могут зависеть миллиарды.

Каким бы ни был ваш путь в мире Биткойна, мы благодарим вас за то, что вы выбрали книгу «Осваиваем Биткойн» в качестве части этого пути.

Приложение A

Техническое описание Биткойна от Сатоши Накамото

Это оригинальный документ, полностью воспроизведенный в том виде, в котором он был опубликован Сатоши Накамото (Satoshi Nakamoto) в октябре 2008 года.

Биткойн – одноранговая система электронных денег

Сатоши Накамото
satoshin@gmx.com
www.bitcoin.org, https://bitcoin.org/en/

Аннотация. Исключительно одноранговая версия электронных денег позволила бы отправлять онлайн-платежи напрямую от одной стороны к другой без посредничества финансового учреждения. Частично этот вопрос решается с помощью цифровых подписей, но основные преимущества будут потеряны, если для предотвращения двойного расходования средств по-прежнему требуется третья доверенная сторона. Мы предлагаем решение проблемы двойного расходования средств с помощью одноранговой сети. Сеть регистрирует транзакции по времени, хешируя их в непрерывной цепочке доказательства работы на основе хеша, формируя запись, которую невозможно изменить без повторного выполнения доказательства работы. Самая длинная цепочка служит не только доказательством наблюдаемой последовательности событий, но и подтверждением того, что она исходит от самого большого пула процессорных мощностей. Пока большая часть процессорной мощности контролируется узлами, которые не взаимодействуют между собой для атаки на сеть, они будут создавать самую длинную цепочку и опережать злоумышленников. Структура самой сети сведена к минимуму. Сообщения передаются по принципу наименьших усилий, а узлы могут покидать сеть и возвращаться в нее по своему

усмотрению, принимая самую длинную цепочку доказательств работы в качестве подтверждения событий, имевших место во время их отсутствия.

Введение

Торговля в интернете практически полностью зависит от финансовых учреждений, выступающих в качестве доверенной стороны при проведении электронных платежей. Хотя эта система достаточно эффективна для большинства транзакций, она все же подвержена слабостям модели на основе доверия. В реальности полностью необратимые транзакции невозможны, поскольку финансовые учреждения не могут избежать посредничества в спорах. Стоимость посредничества увеличивает транзакционные издержки, ограничивая минимальный практический размер транзакции и лишая возможности совершать мелкие случайные сделки, а более масштабные издержки связаны с потерей возможности совершать невозвратные платежи за необратимые услуги. С появлением возможности обратимых платежей возрастает потребность в доверии. Продавцы должны настороженно относиться к своим клиентам, выпытывая у них дополнительную информацию сверх необходимого. Определенный процент мошенничества принимается как неизбежность. Этих издержек и неопределенности при оплате можно избежать с помощью физической валюты, но не существует механизма для осуществления платежей по каналу связи без доверенной стороны.

Что действительно необходимо, так это электронная платежная система, основанная на криптографических подтверждениях, а не на доверии, которая бы позволила двум желающим сторонам совершать сделки напрямую друг с другом без необходимости в доверенной третьей стороне. Транзакции, которые невозможно отменить с помощью вычислений, защитят продавцов от мошенничества, а для защиты покупателей можно с легкостью внедрить обычные механизмы депонирования. В этой статье мы предлагаем решение проблемы двойного расходования средств с помощью одноранговой распределенного сервера временных меток для создания вычислительных доказательств хронологического порядка транзакций. Система безопасна до тех пор, пока добросовестные узлы в совокупности контролируют больше процессорной мощности, чем любая взаимодействующая группа узлов-злоумышленников.

Транзакции

Мы определяем электронную монету как цепочку цифровых подписей. Каждый владелец передает монету следующему, подписывая хеш предыдущей транзакции и открытый ключ следующего владельца цифровой подписью и добавляя их в конец монеты. Получатель платежа может проверить подписи для подтверждения цепочки владения.

Проблема, конечно, в том, что получатель не может проверить отсутствие двойного расходования монеты одним из владельцев. Общим решением является введение доверенного центрального органа, или монетного двора, который проверяет каждую транзакцию на предмет двойной траты. После каждой

транзакции монета должна быть возвращена на монетный двор для выпуска новой монеты, и только монеты, выпущенные непосредственно монетным двором, могут быть защищены от двойного расходования. Проблема с этим решением заключается в том, что судьба всей денежной системы зависит от управляющей компании монетного двора, и каждая транзакция должна проходить через нее, как через банк.

Рис. А.1

Нам нужен способ, чтобы получатель денег знал, что предыдущие владельцы не подписывали никаких более ранних транзакций. Для наших целей важна самая ранняя транзакция, поэтому нас не волнуют последующие попытки двойного расходования. Единственным способом подтверждения отсутствия транзакции является информация обо всех транзакциях. В модели с монетным двором он знает обо всех транзакциях и решает, какая из них пришла первой. Чтобы решить эту задачу без доверенной стороны, транзакции должны быть объявлены публично [1], и нам нужна система, позволяющая участникам договориться о единой истории порядка их получения. Получателю нужно доказательство того, что во время каждой транзакции большинство узлов согласились с тем, что она была получена первой.

Сервер временных меток

Предлагаемое нами решение начинается с сервера временных меток. Для работы сервера временных меток берется хеш блока элементов, которые должны получить временную метку, и публикуется в широком доступе, например в газете или сообщении Usenet [2-5]. Временная метка доказывает, что данные, очевидно, должны были существовать в то время, чтобы попасть в хеш. Каждая временная метка включает в свой хеш предыдущую, образуя цепочку, в которой каждая дополнительная временная метка закрепляет предыдущие.

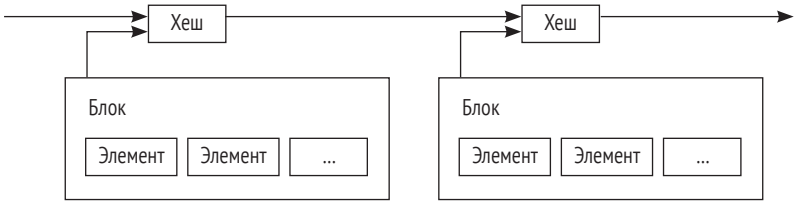


Рис. А.2

Доказательство работы

Чтобы реализовать распределенный сервер временных меток на одноранговой основе, нам потребуется использовать не сообщения в газетах или Usenet, а систему доказательств выполнения работы, подобную Hashcash Адама Бэка (Adam Back) [6]. Доказательство работы заключается в сканировании значения, которое при хешировании, например в SHA-256, начинается с нулевого числа битов. В среднем требуемая работа экспоненциальна числу используемых нулевых битов и может быть проверена с помощью выполнения одного хеша. Для нашей сети временных меток мы реализуем доказательство работы, увеличивая значение нонса в блоке до тех пор, пока не будет найдено значение, дающее хешу блока требуемые нулевые биты. После того как процессор затратил усилия на удовлетворение требования доказательства работы, блок не может быть изменен без повторного выполнения такой работы. Поскольку после него в цепочку выстраиваются последующие блоки, работа по изменению блока будет включать в себя переделку всех следующих за ним блоков.

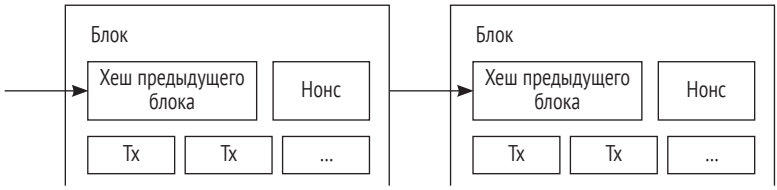


Рис. А.3

Доказательство работы также решает проблему определения репрезентативности при принятии решений большинством голосов. Если бы большинство основывалось на принципе «один IP-адрес – один голос», его мог бы обойти любой, кто способен выделить много IP-адресов. Доказательство работы – это, по сути, «один процессор – один голос». Решение большинства представлено самой длинной цепочкой, в которую вложено наибольшее количество усилий по доказательству работы. Если большинство процессорных мощностей контролируются добросовестными узлами, то честная цепочка будет расти быстрее всех и обгонит все конкурирующие цепочки. Чтобы изменить прошедший блок, злоумышленнику придется переделать доказательство работы этого блока и всех последующих блоков, а затем догнать и превзойти работу добросовестных узлов. Позже мы покажем, что вероятность того, что более медленный

злоумышленник догонит его, экспоненциально уменьшается по мере добавления последующих блоков.

Чтобы компенсировать рост скорости аппаратного обеспечения и изменение интереса к работе узлов с течением времени, сложность доказательства работы определяется скользящим усредненным значением, ориентированным на среднее количество блоков в час. Если они генерируются слишком быстро, сложность увеличивается.

Сеть

Запуск сети осуществляется следующим образом.

1. Новые транзакции рассылаются всем узлам.
2. Каждый узел собирает новые транзакции в блок.
3. Каждый узел работает над поиском сложного доказательства работы для своего блока.
4. Когда узел находит доказательство работы, он рассылает блок всем узлам.
5. Узлы принимают блок только в том случае, если все транзакции в нем действительны и еще не потрачены.
6. Узлы выражают свое согласие с блоком, работая над созданием следующего блока в цепочке, с использованием хеша принятого блока в качестве предыдущего хеша.

Узлы всегда рассматривают самую длинную цепочку как правильную и продолжают работать над ее продлением. Если два узла одновременно передают разные версии следующего блока, некоторые узлы могут получить ту или иную версию первыми. В этом случае они работают над первой полученной версией, но сохраняют другую ветвь на случай, если она станет длиннее. Ничья будет нарушена после получения следующего доказательства работы, и одна из ветвей станет длиннее; узлы, работавшие над другой ветвью, переключатся на более длинную.

Рассылка новых транзакций не обязательно должна достигать всех узлов. Если они доходят до многих узлов, то вскоре попадают в блок. Передачи блоков также толерантны к пропущенным сообщениям. Если узел не получил блок, он запросит его, когда получит следующий блок и поймет, что уже пропустил один.

Стимулирование

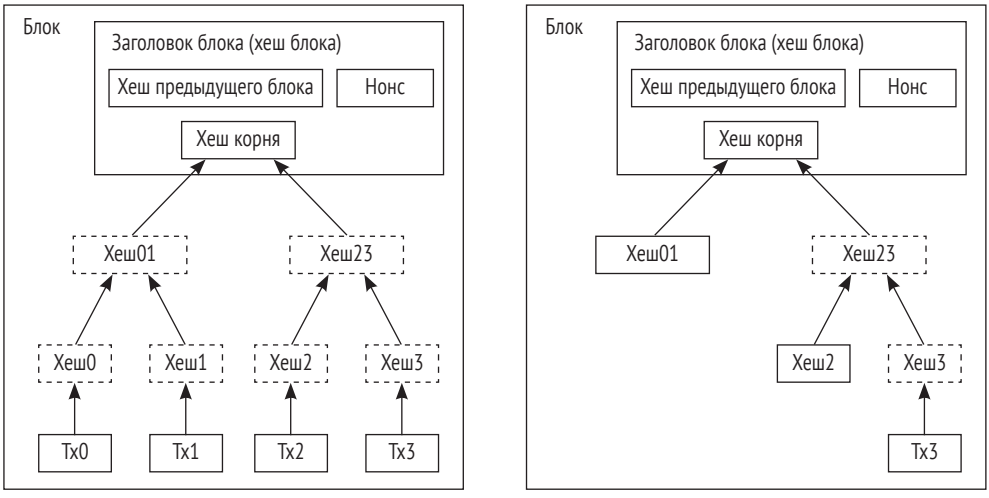
По традиции, первой в блоке является специальная транзакция, которая запускает новую монету, принадлежащую создателю блока. Это дает узлам стимул поддерживать сеть и обеспечивает способ первоначального распространения монет в обращении ввиду отсутствия центрального органа, который бы их выпускал. Постоянное добавление постоянного количества новых монет аналогично тому, как золотодобытчики тратят ресурсы на добавление золота в оборот. В нашем случае это время процессора и электроэнергия.

Стимул также может финансироваться за счет комиссии за транзакции. Когда выходная стоимость транзакции меньше ее входной стоимости, разница составляет комиссию за транзакцию, добавляемую к стимулирующей стоимости блока с транзакцией. После ввода в обращение заранее определенного количества монет стимул может полностью перейти на комиссию за транзакции и стать полностью свободным от инфляции.

Стимул может побуждать узлы оставаться честными. Если жадный злоумышленник сможет собрать больше процессорных мощностей, чем все добросовестные узлы, ему придется выбирать: использовать их для обмана людей, похищая свои платежи, или использовать их для генерации новых монет. Ему будет выгоднее играть по правилам, выгодным для него, получая больше новых монет, чем все остальные вместе взятые, чем подрывать систему и ценность своего собственного капитала.

Использование дискового пространства

Как только последняя транзакция в монете оказывается погребена под достаточным количеством блоков, транзакции, проведенные до нее, могут быть исключены для экономии дискового пространства. Чтобы облегчить эту задачу и не нарушить хеш блока, транзакции хешируются в дерево Меркла [7] [2] [5], при этом в хеш блока включается только корень. Старые блоки могут быть уплотнены путем удаления ветвей дерева. Внутренние хеши хранить не нужно.



Транзакции хешируются в дерево Меркла

После запуска Tx0–2 из блока

Рис. А.4

Заголовок блока без транзакций занимает около 80 байт. Если предположить, что блоки генерируются каждые 10 минут, то 80 байт × 6 × 24 × 365 = 4,2 Мбайта в год. Учитывая, что по состоянию на 2008 год компьютерные системы обычно продаются с 2 Гбайтами оперативной памяти, а закон Мура предсказывает те-

кущий рост на 1,2 Гбайта в год, хранение не должно быть проблемой, даже если заголовки блоков должны храниться в памяти.

Упрощенная верификация платежей

Можно проверять платежи без запуска полного сетевого узла. Пользователю достаточно хранить копию заголовков блоков самой длинной цепочки доказательства работы, которую он может получить, опрашивая узлы сети, пока не убедится, что у него самая длинная цепочка, и получить ветвь Меркла, связывающую транзакцию с блоком, в котором она была отмечена по времени. Он не может проверить транзакцию самостоятельно, но, связав ее с местом в цепочке, он может увидеть, что узел сети принял ее, а добавленные после нее блоки еще больше подтверждают, что сеть ее приняла.

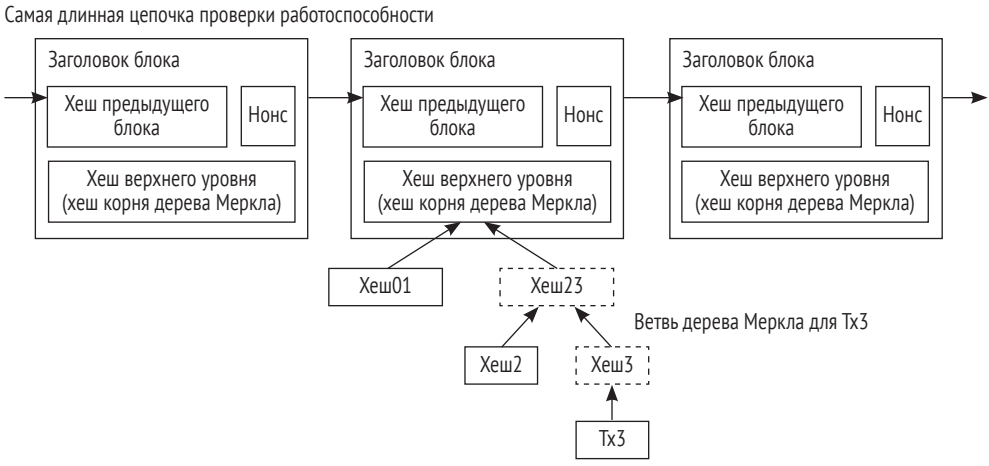


Рис. А.5

Таким образом, проверка является надежной до тех пор, пока добросовестные узлы контролируют сеть, но более уязвимой, если сеть контролирует злоумышленник. Хотя узлы сети могут самостоятельно проверять транзакции, упрощенный метод можно обманывать с помощью сфабрикованных злоумышленником транзакций до тех пор, пока злоумышленник может продолжать контролировать сеть. Одной из стратегий защиты от этого может стать прием предупреждений от узлов сети при обнаружении недействительного блока, что дает возможность ПО пользователя загрузить полный блок и подтвердить несоответствие транзакций с помощью предупреждения. Компании, получающие регулярные платежи, вероятно, все же захотят запустить собственные узлы для обеспечения более независимой защиты и более быстрой проверки.

Объединение и разделение стоимости

Хотя можно было бы работать с монетами по отдельности, проведение отдельной транзакции для каждого цента в переводе было бы громоздким. Для разде-

ления и объединения стоимости транзакции содержат несколько входов и выходов. Обычно это либо один вход из более крупной предыдущей транзакции, либо несколько входов, объединяющих более мелкие суммы, и не более двух выходов: один для оплаты и один для возврата сдачи, если таковая имеется, обратно отправителю.

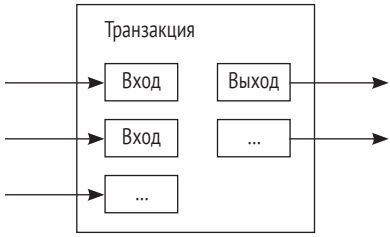


Рис. А.6

Следует отметить, что проблемы веерного распространения, когда транзакция зависит от нескольких транзакций, а те зависят от многих других, здесь не существует. Никогда не возникает необходимости извлекать полную отдельную копию истории транзакции.

Конфиденциальность

В традиционной банковской модели уровень конфиденциальности достигается за счет ограничения доступа к информации только участвующими сторонами и доверенной третьей стороной. Необходимость публично объявлять обо всех транзакциях исключает этот метод, но конфиденциальность все же можно сохранить, прервав поток информации в другом месте: сохранив анонимность открытых ключей. Пользователи могут видеть, что кто-то отправляет определенную сумму другому, но без информации, связывающей транзакцию с кем-либо. Это похоже на уровень информации, предоставляемой фондовыми биржами, где время и размер отдельных сделок, «лента», публикуется открыто, но без указания сторон.

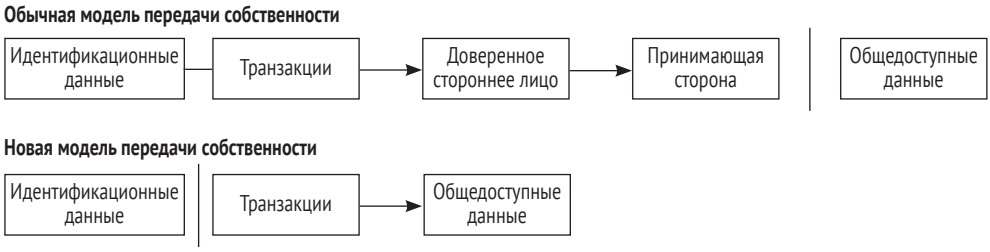


Рис. А.7

В качестве дополнительного средства защиты для каждой транзакции должна использоваться новая пара ключей, чтобы они не были связаны с общим

владельцем. Некоторое связывание все же неизбежно при использовании транзакций с несколькими входами, которые обязательно покажут, что их входы принадлежали одному владельцу. Риск заключается в том, что если владелец ключа будет раскрыт, связывание может выявить другие транзакции, принадлежащие тому же владельцу.

Вычисления

Мы рассматриваем сценарий, в котором злоумышленник пытается сгенерировать альтернативную цепочку быстрее, чем создается истинная цепочка. Даже если это удастся, это не сделает систему открытой для произвольных изменений, таких как создание ценности из воздуха или получение денег, которые никогда не принадлежали злоумышленнику. Узлы не примут недействительную транзакцию в качестве оплаты, а добросовестные узлы никогда не примут блок с ними. Злоумышленник может лишь попытаться изменить одну из своих собственных транзакций, чтобы забрать назад недавно потраченные деньги.

Состязание между добросовестной цепочкой и цепочкой злоумышленника можно охарактеризовать как биномиальное случайное блуждание (Binomial Random Walk). Событием успеха является удлинение добросовестной цепочки на один блок, что увеличивает ее отрыв на +1, а событием неудачи служит удлинение цепочки злоумышленника на один блок, что сокращает разрыв на –1.

Вероятность того, что злоумышленник догонит соперника с заданным дефицитом, аналогична «проблеме разорения азартного игрока». Предположим, что азартный игрок с неограниченным кредитом начинает в минусе и разыгрывает потенциально бесконечное число партий, пытаясь выйти в плюс. Вероятность того, что он когда-нибудь достигнет уровня прибыльности или что злоумышленник когда-нибудь догонит честную цепочку, можно вычислить следующим образом [8]:

$$q_z = \begin{cases} 1, & \text{если } p \leqslant q \\ (q/p)^2, & \text{если } p > q \end{cases},$$

где

- p – вероятность того, что добросовестный узел найдет следующий блок;
- q – вероятность того, что злоумышленник найдет следующий блок;
- q_z – вероятность того, что злоумышленник когда-либо догонит цепочку с отставанием в z блоков.

С учетом нашего предположения, что $p > q$, вероятность падает экспоненциально по мере увеличения количества блоков, которые злоумышленник должен догнать. Если он не сделает удачный рывок вперед на ранней стадии, его шансы станут исчезающе малы по мере дальнейшего отставания.

Теперь мы рассмотрим, сколько времени нужно ждать получателю новой транзакции, чтобы быть достаточно уверенным в том, что отправитель не может изменить транзакцию. Мы предполагаем, что отправителем является злоумышленник, который хочет заставить получателя поверить, что он заплатил ему на время, а затем по истечении некоторого времени сделать подмену,

чтобы вернуть свой платеж. Получатель будет предупрежден, когда это произойдет, но отправитель надеется, что будет слишком поздно.

Получатель генерирует новую пару ключей и передает открытый ключ отправителю незадолго до подписания. Это не позволяет отправителю заранее подготовить цепочку блоков, непрерывно работая над ней до момента, когда ему посчастливится продвинуться достаточно далеко, и в этот момент провести транзакцию. Как только транзакция отправлена, мошенник-отправитель начинает тайно работать над параллельной цепочкой, содержащей альтернативную версию его транзакции.

Получатель ждет, пока транзакция не будет добавлена в блок и после нее не будут подключены z блоков. Он не знает точной скорости работы злоумышленника, но если предположить, что на создание честно исполненных блоков ушло среднее ожидаемое время, то потенциальная скорость работы злоумышленника будет представлять собой распределение Пуассона с ожидаемым значением:

$$\lambda = z\frac{q}{p}.$$

Чтобы определить вероятность того, что злоумышленник все еще может догнать нас сейчас, умножим распределение Пуассона для каждого уровня прогресса, которого он мог бы достичь, на вероятность того, что он может догнать его с этой точки:

$$\sum_{k=0}^{\infty}\frac{\lambda^k e^{-\lambda}}{k!}\cdot\begin{cases}(q/p)^{(z-k)}, & \text{если } k \leqslant z\\ 1, & \text{если } k > z\end{cases}.$$

Перегруппировка с целью избежать суммирования бесконечного хвоста распределения...

$$1-\sum_{k=0}^{z}\frac{\lambda^k e^{-\lambda}}{k!}\left(1-(q/p)^{(z-k)}\right).$$

Преобразование в код на языке C...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Получив некоторые результаты, мы видим, что вероятность экспоненциально убывает с ростом z.

```
q=0.1
z=0 P=1.0000000
z=1 P=0.2045873
z=2 P=0.0509779
z=3 P=0.0131722
z=4 P=0.0034552
z=5 P=0.0009137
z=6 P=0.0002428
z=7 P=0.0000647
z=8 P=0.0000173
z=9 P=0.0000046
z=10 P=0.0000012
q=0.3
z=0 P=1.0000000
z=5 P=0.1773523
z=10 P=0.0416605
z=15 P=0.0101008
z=20 P=0.0024804
z=25 P=0.0006132
z=30 P=0.0001522
z=35 P=0.0000379
z=40 P=0.0000095
z=45 P=0.0000024
z=50 P=0.0000006
Solving for P less than 0.1%…
P < 0.001
q=0.10 z=5
q=0.15 z=8
q=0.20 z=11
q=0.25 z=15
q=0.30 z=24
q=0.35 z=41
q=0.40 z=89
q=0.45 z=340
```

Заключение

Мы предложили систему для электронных транзакций без зависимости от доверия. Мы начали с обычной системы монет на основе цифровых подписей, которая обеспечивает надежный контроль за владением, но является неполной без способа предотвращения двойных трат. Чтобы решить эту проблему, мы предложили одноранговую сеть, использующую доказательство работы для записи публичной истории транзакций, которая быстро становится вычислительно невыполнимой для злоумышленника, если добросовестные узлы контролируют большую часть мощности процессора. Сеть надежна в своей неструктурированной простоте. Узлы работают все сразу и практически не координируются. Их не нужно идентифицировать, поскольку сообщения не направляются в какое-либо конкретное место и должны доставляться только по принципу наибольшей эффективности. Узлы могут выходить из сети и воз-

вращаться в нее по своему желанию, принимая цепочку доказательств работы в качестве подтверждения того, что произошло, пока их не было. Они голосуют своими процессорными мощностями, выражая свое согласие с действительными блоками, работая над их расширением и отвергая недействительные блоки, отказываясь работать над ними. Любые необходимые правила и стимулы могут быть реализованы с помощью этого механизма консенсуса.

Список литературы

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.
[2] H. Massias, X. S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux, May 1999.
[3] S. Haber, W. S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99–111, 1991.
[4] D. Bayer, S. Haber, W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in Communication, Security and Computer Science, pages 329–334, 1993.
[5] S. Haber, W. S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, pages 28–35, April 1997.
[6] A. Back, "Hashcash – a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.
[7] R. C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122–133, April 1980.
[8] W. Feller, "An introduction to probability theory and its applications," 1957.

Лицензия

Этот документ был опубликован в октябре 2008 года от имени Сатоши Накамото. Позднее (в 2009 году) он был добавлен в качестве сопроводительной документации к программному обеспечению биткойна и распространяется под той же лицензией Массачусетского технологического института (МТИ). Она воспроизведена в этой книге без изменений, за исключением форматирования, на условиях лицензии МТИ:

Указанное выше уведомление об авторском праве и данные условия должны быть включены во все копии или значимые части данного Программного обеспечения.

ДАННОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРЕДОСТАВЛЯЕТСЯ «КАК ЕСТЬ», БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, ЯВНО ВЫРАЖЕННЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ГАРАНТИЯМИ ТОВАРНОЙ ПРИГОДНОСТИ, СООТВЕТСТВИЯ ПО ЕГО КОНКРЕТНОМУ НАЗНАЧЕНИЮ И ОТСУТСТВИЯ НАРУШЕНИЙ ПРАВ. НИ В КАКОМ СЛУЧАЕ АВТОРЫ ИЛИ ПРАВООБЛАДАТЕЛИ НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ПО ИСКАМ О ВОЗМЕЩЕНИИ УЩЕРБА, УБЫТКОВ ИЛИ ДРУГИХ ТРЕБОВАНИЙ ПО ДЕЙСТВУЮЩИМ КОНТРАКТАМ, ДЕЛИКТАМ ИЛИ ИНОМУ, ВОЗНИКШИМ ИЗ ИМЕЮЩИХ ПРИЧИНОЙ ИЛИ СВЯЗАННЫХ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ ИЛИ ИСПОЛЬЗОВАНИЕМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИЛИ ИНЫХ ДЕЙСТВИЙ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ.

Приложение B

Ошибки в техническом описании Биткойна

В этом приложении содержится описание известных проблем в работе Сатоши Накамото «Биткойн: одноранговая система электронных денег» (Bitcoin: A Peer-to-Peer Electronic Cash System), а также примечания об изменениях в терминологии и о том, чем реализация Биткойна отличается от описанной в работе.

Этот документ был первоначально опубликован одним из соавторов данной книги в 2016 году; он воспроизводится здесь с обновлениями. Названия разделов в этом исправлении соответствуют названиям разделов в оригинальной статье Накамото.

Аннотация

> «Самая длинная цепочка служит не только доказательством наблюдаемой последовательности событий, но и подтверждением того, что она исходит от самого большого пула процессорных мощностей».

Детали реализации. Если бы каждое звено цепи (в Биткойне они называются «блоками») строилось с использованием одинакового количества доказательств работы (proof of work, PoW), то самой длинной цепью была бы та, которая опирается на самый большой пул вычислительных мощностей. Однако Биткойн был реализован таким образом, что количество PoW может варьироваться между блоками, поэтому стало важно проверять не «самую длинную цепочку», а «цепочку с наибольшим количеством PoW»; это часто сокращается до «самой работоспособной цепочки».

Переход от проверки самой длинной цепочки к проверке самой рабочей цепочки произошел в июле 2010 года, спустя длительное время после первоначального выпуска Биткойна:

```
- if (pindexNew->nHeight > nBestHeight)
+ if (pindexNew->bnChainWork > bnBestChainWork)
```

Изменение терминологии. Для генерирования PoW первых блоков Биткойна использовались обычные центральные процессоры, но сегодня генерация PoW в основном осуществляется специализированными прикладными интегральными схемами (ASIC), поэтому вместо слова «мощность процессора», возможно, правильнее говорить «вычислительная мощность» или, проще говоря, «хешрейт» для хеширования, используемого при генерации PoW.

> «Пока большая часть процессорной мощности контролируется узлами, которые не взаимодействуют между собой для атаки на сеть, они будут создавать самую длинную цепочку и опережать злоумышленников».

Изменение терминологии. Термин «узлы» сегодня используется для обозначения узлов полной валидации, которые представляют собой программы с соблюдением всех правил системы. Программы (и аппаратные средства), которые увеличивают цепочку, сегодня называются «майнерами», исходя из аналогии Накамото с золотодобытчиками, приведенной в разделе 6 статьи. Накамото ожидал, что все майнеры будут узлами, но выпущенное им ПО не требовало, чтобы все узлы были майнерами. В оригинальном ПО простой пункт меню в графическом интерфейсе узла позволял включить или выключить функцию майнинга.

Сегодня подавляющее число узлов не являются майнерами, и многие люди, владеющие оборудованием для майнинга, не используют его на своих узлах (и даже те, кто майнит на своих узлах, часто майнят в течение коротких периодов времени поверх вновь обнаруженных блоков, не убедившись, что их узел считает новый блок действительным). В ранних частях статьи, где слово «узлы» используется без изменений, речь идет о майнинге с помощью узла с полной валидацией; в более поздних частях статьи, где говорится о «сетевых узлах», речь идет в основном о том, что могут делать узлы, даже если они не майнят.

Обнаружено после публикации. Когда добывается новый блок, добывающий его майнер может сразу же начать работу над его продолжением, но все остальные майнеры не знают о новом блоке и не могут начать работу над ним, пока он не распространится по сети. Это дает майнерам, добывающим много блоков, преимущество перед майнерами, добывающими меньше блоков, и это может быть использовано в так называемой атаке эгоистичного майнинга, позволяющей злоумышленнику с примерно 30 % общего хешрейта сети сделать других майнеров менее прибыльными, возможно, заставив их следовать политике атакующего майнера. Поэтому вместо утверждения «пока большая часть процессорной мощности контролируется узлами, которые не взаимодействуют между собой для атаки на сеть», возможно, правильнее будет сказать «пока узлы, сотрудничающие в атаках на сеть, контролируют менее 30 % сети».

Транзакции

> «Мы определяем электронную монету как цепочку цифровых подписей. Каждый владелец передает монету следующему, подписывая хеш предыдущей транзакции и открытый ключ следующего владельца цифровой подписью и добавляя их в конец монеты».

Детали реализации. В Биткойне реализована более общая версия этой системы, в которой цифровые подписи не используются напрямую, а вместо них применяется «детерминированное выражение». Точно так же, как подпись, соответствующая известному открытому ключу, может быть использована для осуществления платежа, данные, удовлетворяющие известному выражению, тоже могут обеспечить платеж. В общем случае выражение, которое должно быть выполнено в Биткойне для расходования монеты, называют «обременение» (encumbrance). Почти все обременения в Биткойне на сегодняшний день требуют предоставления хотя бы одной подписи. Поэтому вместо фразы «цепочка цифровых подписей» правильнее говорить «цепочка обременений». Учитывая, что транзакции часто имеют более одного входа и более одного выхода, структура не очень похожа на цепочку; ее правильнее описывать как направленный ациклический граф (directed acyclic graph, DAG).

Доказательство работы

> «…мы реализуем доказательство работы, увеличивая значение нонса в блоке до тех пор, пока не будет найдено значение, дающее хешу блока требуемые нулевые биты».

Детали реализации. Реализация Hashcash Адама Бэка требует найти хеш с необходимым количеством ведущих нулевых битов. Биткойн рассматривает хеш как целое число и требует, чтобы оно было меньше заданного целого числа, что фактически позволяет указывать дробное количество битов.

> «Доказательство работы – это, по сути, “один процессор – один голос”».

Важное замечание. Голосование здесь происходит не по правилам системы, а лишь по порядку транзакций, чтобы обеспечить гарантии невозможности двойной траты «электронной монеты». Более подробно это описано в разделе 11 документа, где говорится: «Мы рассматриваем сценарий, в котором злоумышленник пытается сгенерировать альтернативную цепочку быстрее, чем создается истинная цепочка. Даже если это удастся, это не сделает систему открытой для произвольных изменений, таких как создание ценности из воздуха или получение денег, которые никогда не принадлежали злоумышленнику. Узлы не примут недействительную транзакцию в качестве оплаты, а добросовестные узлы никогда не примут блок с ними».

> «…сложность доказательства работы определяется скользящим усредненным значением, ориентированным на среднее количество блоков в час».

Детали реализации. Скользящее усредненное значение не используется. Вместо этого время генерации каждого 2016-го блока сравнивается со временем генерации более раннего блока, и разница между ними используется для расчета среднего значения, применяемого для корректировки.

Кроме того, среднее значение, реализованное в Биткойне, нацелено на среднее количество блоков в две недели (а не в час, как может подразумеваться в тексте). Другие реализованные правила могут еще больше замедлить коррек-

тировку, например правило, согласно которому корректировка не может увеличить скорость майнинга блоков более чем на 300 % за период или замедлить ее более чем на 75 %.

Использование дискового пространвства

> «Как только последняя транзакция в монете оказывается погребена под достаточным количеством блоков, транзакции, проведенные до нее, могут быть исключены для экономии дискового пространства».

Возможное выявление после публикации. Хотя структура дерева Меркла, описанная в этом разделе, может доказать включение транзакции в конкретный блок, в настоящее время в Биткойне не существует способа доказать, что транзакция не была потрачена, кроме обработки всех последующих данных в блокчейне. Это означает, что описанный здесь метод не может быть универсально использован для освобождения дискового пространства между всеми узлами, поскольку все новые узлы должны будут обрабатывать все транзакции.

Упрощенная верификация платежей

> «Одной из стратегий защиты от этого может стать прием предупреждений от узлов сети при обнаружении недействительного блока, что дает возможность ПО пользователя загрузить полный блок и подтвердить несоответствие транзакций с помощью предупреждения».

Важное замечание. Хотя было создано ПО, реализующее некоторые части этого раздела и называющее его упрощенной верификацией платежей (Simplified Payment Verification, SPV), ни одна из этих программ в настоящее время не принимает предупреждения от узлов сети (узлов полной валидации) об обнаружении недействительных блоков. Из-за этого в прошлом биткойны в так называемых SPV-кошельках подвергались риску.

Конфиденциальность

> «Некоторое связывание все же неизбежно при использовании транзакций с несколькими входами, которые обязательно покажут, что их входы принадлежали одному владельцу».

Выявлено после публикации. Не совсем понятно, что у разных входов одной и той же транзакции один и тот же владелец, если владельцы часто смешивают свои входы с входами, принадлежащими другим владельцам. Например, нет никакой публичной разницы между тем, что Алиса и Боб внесли по одному

своему входу в оплату Чарли и Дэна, и тем, что просто Алиса внесла два своих входа в оплату Чарли и Дэна.

Сегодня эта техника известна под названием CoinJoin, а ПО, реализующее ее, используется с 2015 года.

Вычисления

> «Получатель генерирует новую пару ключей и передает открытый ключ отправителю незадолго до подписания. Это не позволяет отправителю заранее подготовить цепочку блоков, непрерывно работая над ней до момента, когда ему посчастливится продвинуться достаточно далеко и в этот момент провести транзакцию».

Обнаружено после публикации. Ничто в том, что получатель генерирует открытый ключ незадолго до подписания транзакции отправителем, не мешает ему подготовить цепочку блоков заранее. Ранний пользователь Биткойна Хэл Финни (Hal Finney) обнаружил эту атаку и описал (https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384) ее так: «Предположим, что злоумышленник периодически генерирует блоки. В каждый сгенерированный блок он включает перевод с адреса A на адрес B, оба из которых он контролирует. Чтобы обмануть вас при генерации блока, он не отправляет его по сети. Вместо этого он бежит в ваш магазин и совершает платеж на ваш адрес C со своего адреса A. Вы ждете несколько секунд, ничего не замечаете и отправляете товар. Теперь он передает свой блок, и его транзакция будет иметь приоритет над вашей».

Атака работает при любом количестве подтверждений и иногда называется атакой Финни (Finney Attack).

Отказ от ответственности. Автор этого документа не был первым, кто выявил какую-либо из описанных здесь проблем, он просто собрал их в один документ.

Обновления, сделанные после публикации этой книги, см. в Оригинальном документе https://gist.github.com/harding/dabea3d83c695e6b937bf090eddf2bb3.

Приложение C

Предложения по улучшению Биткойна

Предложения по улучшению Биткойна (Bitcoin Improvement Proposals, BIP) относятся к проектным документам, предоставляющим информацию сообществу Биткойна или описывающим новую функцию для Биткойна, его процессов или среды.

В соответствии с «Целями и рекомендациями в BIP» (BIP Purpose and Guidelines) протокола BIP1 существует три вида BIP:

Стандартный BIP
: Описывает любое изменение, которое затрагивает большинство или все реализации Биткойна, например изменение сетевого протокола, изменение правил валидности блоков или транзакций либо любое изменение или дополнение, которое влияет на совместимость приложений, использующих Биткойн.

Информационный BIP
: Описывает проблемы разработки Биткойна или предоставляет общие рекомендации либо информацию сообществу Биткойна, но не предлагает новую функцию. Информационные BIP не обязательно представляют собой консенсус или рекомендацию сообщества Биткойна, поэтому пользователи и разработчики могут игнорировать информационные BIP или не следовать их советам.

Технологический (*Process*) *BIP*
: Описывает процессы в Биткойне либо предлагает изменить процесс (или событие в нем). Процессовые BIP похожи на стандартные BIP, но относятся не к самому протоколу Биткойна, а к другим аспектам. Они могут предлагать реализацию, но не в кодовой базе самого Биткойна; часто они требуют консенсуса сообщества. В отличие от информационных BIP, они представляют собой нечто большее, чем рекомендации, и пользователи, как правило, не могут их игнорировать. В качестве примера можно привести процедуры, руководства, изменения в процессе принятия решений, а также изменения в инструментах или среде, используемых при разработке Биткойна.

Любой мета-BIP также считается BIP процесса.

Протоколы BIP записываются в репозиторий версий на GitHub. В документе проекта Bitcoin Core с открытым исходным кодом под лицензией МИТ, который воспроизводится здесь в отредактированном виде, описано, какие BIP в нем реализованы, включая список Pull Request (PR) и версию Bitcoin Core, в которой была добавлена или существенно изменена поддержка каждого BIP.

Список BIP, реализованных в Bitcoin Core:

- BIP9: изменения, позволяющие параллельно развертывать несколько софт-форков, были реализованы начиная с версии 0.12.1 (PR #7575);
- BIP11: выходы мультиподписи стали стандартными с версии 0.6.0 (PR #669);
- BIP13: формат адресов P2SH реализован с версии 0.6.0 (PR #669);
- BIP14: строка subversion используется в качестве User Agent с версии 0.6.0 (PR #669);
- BIP16: правила оценки pay-to-script-hash были реализованы с версии 0.6.0 и вступили в силу 1 апреля 2012 года (PR #748);
- BIP21: формат URI для платежей в биткойнах был реализован с версии 0.6.0 (PR #176);
- BIP21: формат URI для платежей в биткойнах был реализован с версии 0.6.0 (PR #176);
- BIP22: RPC-протокол *getblocktemplate* (GBT) для майнинга был реализован начиная с версии 0.7.0 (PR #936);
- BIP23: некоторые расширения GBT были реализованы с версии 0.10.0rc1, включая longpolling и предложения блоков (PR #1816);
- BIP30: правила оценки, запрещающие создавать новые транзакции с тем же txid, что и предыдущие не полностью проведенные транзакции, были реализованы начиная с версии 0.6.0, и правило вступило в силу 15 марта 2012 года (PR #915);
- BIP31: сообщение протокола *pong* (и увеличение версии протокола до 60001) было реализовано с версии 0.6.1 (PR #1081);
- BIP32: иерархические детерминированные кошельки были реализованы начиная с версии 0.13.0 (PR #8035);
- BIP34: правило, требующее, чтобы блоки содержали свою высоту (номер) во входных данных coinbase и введение блоков версии 2, было реализовано с v0.7.0. Правило вступило в силу для блоков версии 2 с блока 224 413 (5 марта 2013), а блоки версии 1 больше не допускаются с блока 227 931 (25 марта 2013) (PR #1526);
- BIP35: сообщение протокола *mempool* (и повышение версии протокола до 60002) было реализовано с версии 0.7.0 (PR #1641). Начиная с версии 0.13.0 это доступно только для пиров `NODE_BLOOM` (BIP111);
- BIP37: фильтр Блума для пересылки транзакций, частичные деревья Меркла для блоков и изменение версии протокола до 70001 (для облегченных клиентов с низкой пропускной способностью) были реализова-

ны начиная с версии 0.8.0 (PR #1795). Отключено по умолчанию с v0.19.0, может быть включено опцией `–peerbloomfilters`;

- BIP42: ошибка, из-за которой график субсидий возобновлялся после блока 1 3440 000, была исправлена в v0.9.2 (PR #3842);
- BIP43: экспериментальные дескрипторные кошельки, представленные в v0.21.0, по умолчанию используют деривацию иерархического детерминированного кошелька, предложенную в BIP43 (PR #16528);
- BIP44: экспериментальные дескрипторные кошельки, представленные в версии 0.21.0, по умолчанию используют производную Hierarchical Deterministic Wallet, предложенную в BIP44 (PR #16528);
- BIP49: экспериментальные дескрипторные кошельки, представленные в версии 0.21.0, по умолчанию используют производную Hierarchical Deterministic Wallet, предложенную в BIP49 (PR #16528);
- BIP61: сообщение об отказе от протокола (и увеличение версии протокола до 70 002) было добавлено в v0.9.0 (PR #3185). Начиная с v0.17.0 отправка сообщений об отклонении может быть настроена с помощью опции `-enablebip61`, а в v0.18.0 эта поддержка устарела (отключена по умолчанию). Поддержка была удалена в v0.20.0 (PR #15437);
- BIP65: софт-форк `CHECKLOCKTIMEVERIFY` был объединен в v0.12.0 (PR #6351) и перенесен в v0.11.2 и v0.10.4. `CLTV` только для Mempool был добавлен в PR #6124;
- BIP66: строгие правила DER и связанные с ними блоки версии 3 были реализованы с v0.10.0 (PR #5713);
- BIP68: блокировки последовательности были реализованы начиная с версии 0.12.1 (PR #7184) и похоронены начиная с версии 0.19.0 (PR #16060);
- BIP70 71 72: поддержка протокола платежей доступна в графическом интерфейсе Bitcoin Core начиная с версии 0.9.0 (PR #5216). С версии 0.18.0 (PR 14451) поддержку можно опционально отключить во время сборки, а с версии 0.19.0 (PR #15584) она отключена по умолчанию во время сборки. Она была удалена начиная с версии 0.20.0 (PR 17165);
- BIP84: экспериментальные дескрипторные кошельки, представленные в версии 0.21.0, по умолчанию используют деривацию Hierarchical Deterministic Wallet, предложенную в BIP84 (PR #16528);
- BIP86: дескрипторные кошельки по умолчанию используют деривацию Hierarchical Deterministic Wallet, предложенную в BIP86, начиная с версии 23.0 (PR #22364);
- BIP90: механизм триггеров для активации BIP 34, 65 и 66 был упрощен до проверки высоты блока начиная с версии 0.14.0 (PR #8391);
- BIP111: служебный бит `NODE_BLOOM` добавлен и применяется для всех версий пиров начиная с v0.13.0 (PR #6579 и PR #6641);
- BIP112: операционный код `CHECKSEQUENCEVERIFY` был реализован с версии 0.12.1 (PR #7524) и был закрыт с версии 0.19.0 (PR #16060);

- BIP113: вычисление медианного времени после блокировки было реализовано с версии 0.12.1 (PR #6566), а с версии 0.19.0 (PR #16060) было закрыто;
- BIP125: частично реализована сигнализация о полной замене на комиссию;
- BIP130: прямое объявление заголовков согласовывается с пирами версии ⩾ 70 012 начиная с версии 0.12.0 (PR 6494);
- BIP133: сообщения feefilter соблюдаются и отправляются для пиров версий ⩾ 70 013 начиная с версии 0.13.0 (PR 7542);
- BIP141: Segregated Witness (уровень консенсуса) начиная с версии 0.13.0 (PR 8149), определен для mainnet начиная с версии 0.13.1 (PR 8937) и закрыт начиная с версии 0.19.0 (PR #16060);
- BIP143: проверка подписи транзакций для программы свидетелей версии 0 начиная с версии 0.13.0 (PR 8149), определена для mainnet начиная с версии 0.13.1 (PR 8937) и закрыта начиная с версии 0.19.0 (PR #16060);
- BIP144: обновление Segregated Witness в версии 0.13.0 (PR 8149);
- BIP145: обновление getblocktemplate для Segregated Witness в версии 0.13.0 (PR 8149);
- BIP147: софт-форк `NULLDUMMY` в версии 0.13.1 (PR 8636 и PR 8937), закрыт с версии 0.19.0 (PR #16060);
- BIP152: компактная передача блоков и связанные с ней оптимизации используются с версии 0.13.0 (PR 8068);
- BIP155: сообщения *addrv2* и *sendaddrv2*, позволяющие передавать адреса Tor V3 (и других сетей), поддерживаются начиная с версии 0.21.0 (PR 19954);
- BIP157 158: компактные блокчейн-фильтры для легких клиентов могут индексироваться начиная с версии 0.19.0 (PR #14121) и обслуживаться пирами в P2P-сети начиная с версии 0.21.0 (PR #16442);
- BIP159: служебный бит `NODE_NETWORK_LIMITED` получает сигнал начиная с версии 0.16.0 (PR 11740), а подключение к таким узлам осуществляется начиная с версии 0.17.0 (PR 10387);
- BIP173: адреса Bech32 для собственных выходов Segregated Witness поддерживаются начиная с версии 0.16.0 (PR 11167). Начиная с версии 0.20.0 (PR 16884) адреса Bech32 генерируются по умолчанию;
- BIP174: RPC для работы с частично подписанными транзакциями Биткойна (PSBT) присутствуют в версии 0.17.0 (PR 13557);
- BIP176: деноминация битов [только QT] поддерживается начиная с версии 0.16.0 (PR 12035);
- BIP325: тестовая сеть Signet поддерживается начиная с версии 0.21.0 (PR 18267);
- BIP339: передача транзакций по wtxid поддерживается начиная с версии 0.21.0 (PR 18044);

- BIP340 341 342: правила валидации для Taproot (включая подписи Шнорра и листья Tapscript) реализованы в версии 0.21.0 (PR 19953), а активация в основной сети реализована в версии 0.21.1 (PR 21377, PR 21686);
- BIP350: адреса нативных сегрегированных выходов Witness v1+ используют bech32m вместо bech32 начиная с версии v22.0 (PR 20861);
- BIP371: поля Taproot для PSBT начиная с версии 24.0 (PR 22558);
- BIP380 381 382 383 384 385: дескрипторы скриптов выхода и большинство выражений скриптов реализованы начиная с версии 0.17.0 (PR 13697);
- BIP386: дескрипторы скриптов выхода tr() реализованы в версии 22.0 (PR 22051).

Предметный указатель

В

Г

Д

З

И

К

Р

С

Андреас М. Антонопулос
и Дэвид А. Хардинг

Осваиваем Биткойн

Третье издание

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Бахур В. И.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура PT Serif. Печать офсетная.
Усл. печ. л. 31,36. Тираж 100 экз.

Веб-сайт издательства: **www.dmkpress.com**

O'REILLY

Станьте участником технологической революции, которая захватывает финансовый мир. Данная книга – это ваш проводник по кажущемуся сложным миру биткойна. Создаете ли вы будущее приложение-убийцу, инвестируете в стартап или просто интересуетесь технологией, это переработанное и дополненное издание предоставит вам подробную информацию для начала работы. Биткойн, первая успешная децентрализованная цифровая валюта, уже успел породить многомиллиардную глобальную экономику, которая открыта для всех желающих с достаточными навыками и желанием принять в ней участие. От вас требуется лишь увлеченность.

Третье издание включает:

- подробное знакомство с биткойном и лежащим в его основе блокчейном – идеальный вариант для новичков, инвесторов и руководителей компаний;
- объяснение технических аспектов биткойна и криптовалюты для разработчиков, инженеров, программистов и системных архитекторов;
- подробные сведения о децентрализованной сети Bitcoin, архитектуре одноранговой сети, жизненном цикле транзакций и методах обеспечения безопасности;
- новые разработки, такие как Taproot, Tapscript, подписи Шнорра и Lightning Network;
- глубокое погружение в тему биткойн-приложений, в том числе способов объединения строительных блоков платформы в новые мощные инструменты;
- истории пользователей, аналогии, примеры и фрагменты кода, иллюстрирующие ключевые технические концепции.

«Спустя почти десятилетие после выхода в свет книги «Mastering Bitcoin» третье издание закрепило за ней роль основного источника технического и образовательного контента по теме биткойна. Ни одна другая книга не является столь же всеобъемлющей и актуальной».

Олаолува Осунтокун, Lightning Labs

«Всесторонний обзор всего, что происходит в системе биткойна, и как все это сочетается друг с другом».

Марк «Мурч» Эрхардт, Chaincode Labs

Андреас М. Антонопулос – эксперт по биткойну и открытым блокчейн-технологиям.

Дэвид А. Хардинг – соавтор еженедельного бюллетеня Bitcoin Optech.

Страница книги на dmkpress.com

ISBN 978-6-01810-344-5