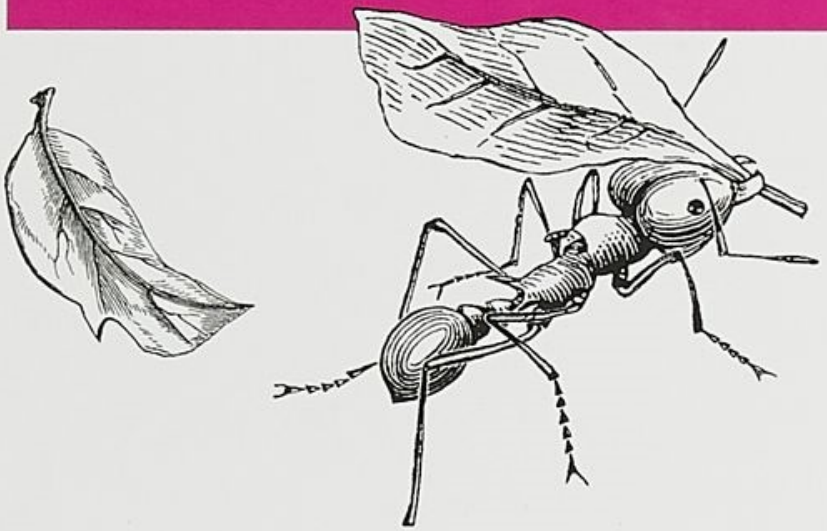
O'REILLY®

Осваиваем биткойн



Андреас М. Антонопулос

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

Андреас М. Антонопулос

Осваиваем биткойн

Andreas M. Antonopoulos

Mastering Bitcoin

Programming the Open Blockchain

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Андреас М. Антонопулос

Осваиваем биткойн

Программирование блокчейна

Москва, 2018

УДК 004.738.5:336.74Bitcoin
ББК 32.971.35+65.262.6с
А72

Антонопулос А. М.
А72 Осваиваем биткойн / пер. с анг. А. В. Снастина. – М.: ДМК Пресс, 2018. – 428 с.: ил.

ISBN 978-5-94074-965-3

Второе издание бестселлера включает подробное введение в самую известную криптовалюту – биткойн, а также в лежащую в ее основе технологию блокчейна. 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 978-1-491-95438-6 (анг.)
ISBN 978-5-94074-965-3 (рус.)

Содержание

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

Предисловие

Как я писал книгу о биткойне

Про биткойн я впервые услышал в середине 2011 года. Первое впечатление было приблизительно таким: «Пфф! Деньги для умников-ботаников», – и я забыл об этом на следующие шесть месяцев, не оценив важности этого явления. Впоследствии подобную реакцию я часто видел у многих умнейших людей, знакомых мне, и это немного утешает. Когда я встретился с биткойном во второй раз при обсуждении в списке рассылки, я решил прочитать документ с техническим описанием, написанный Сатоши Накамото (Satoshi Nakamoto), чтобы изучить авторитетный источник и понять, о чем вообще идет речь. До сих пор помню тот момент, когда я прочитал эти девять страниц, когда осознал, наконец, что биткойн – это не просто цифровые деньги, а сеть доверия, которая могла бы также стать основой для гораздо большего. Осознание того, что «биткойн – это не деньги, а децентрализованная сеть доверия», стало исходным пунктом для четырехмесячного исследования, во время которого я жадно поглощал каждый фрагмент информации о биткойне, который мне попадался. Эта тема овладела моим умом, я полностью увлекся ею, не отходя от компьютера по 12 и более часов в сутки, читал, писал, программировал, изучал все, что мог. Из этого состояния отрешенности от действительности я вышел, похудев на 20 фунтов (около 9 кг) из-за недостаточно полноценного питания, твердо решив вплотную заняться работой с биткойном.

Два года спустя, после создания нескольких небольших стартапов, использующих разнообразные сервисы и продукты, связанные с технологией биткойна, я решил, что пришло время для того, чтобы написать свою первую книгу. Биткойн как неисчерпаемый источник вдохновения занимал все мои мысли, эта технология стала самой значимой со времени появления Интернета. Настало время поделиться моей увлеченностью, моими знаниями об этой великолепной технологии с более широкой аудиторией.

Для кого предназначена эта книга

Эта книга предназначена в основном для программистов-кодеров. Если вы можете писать программы на каком-либо языке программирования, то из этой книги вы узнаете, как работают криптографические валюты, как их использовать и как разрабатывать программное обеспечение (ПО) для работы с ними. Кроме того, несколько первых глав можно рассматривать как подробное введение в технологию биткойна для тех, кто не занимается программированием, но пытается понять внутреннее устройство и функционирование биткойна и криптографических валют.

Почему на обложке изображены насекомые?

Муравей-листорез относится к биологическим видам, демонстрирующим чрезвычайно сложное поведение в колонии социальных насекомых (суперорганизме), но каждый отдельный муравей действует в соответствии с набором простых правил, соответствующих принципам социального взаимодействия и основанных на обмене химическими ароматическими веществами (феромонами). Цитата из Википедии: «Муравьи-листорезы образуют самые крупные и самые сложные сообщества живых организмов на Земле, если не считать людей». В действительности муравьи-листорезы не едят листья, но используют их для разведения особого вида грибов, являющегося основным источником питания для колонии. Вы понимаете? Муравьи занимаются сельскохозяйственным производством!

Несмотря на то что муравьи образуют кастовое сообщество и у них имеется матка-королева для производства потомства, все же у них нет централизованного органа управления или лидера всей муравьиной колонии. Высокий интеллект и разумное поведение, демонстрируемое многомиллионной колонией, является так называемым эмерджентным свойством (emergent property), системным эффектом, возникающим или проявляющимся как следствие взаимодействия отдельных членов социальной сети.

Природа наглядно показывает, что децентрализованные системы могут быть весьма гибкими и проявлять эмерджентную (приобретенную, а не врожденную) сложность и невероятную изощренность поведения без обязательного наличия в них центрального органа управления, иерархии или сложных составных частей.

Биткойн – это чрезвычайно разумная и изощренная децентрализованная сеть доверия, которая может поддерживать огромное количество финансовых процессов. При этом каждый узел сети биткойн следует нескольким простым математическим правилам. Такое взаимодействие множества узлов как раз и приводит к формированию разумного поведения при отсутствии, казалось бы, неизбежной сложности или доверия к отдельно взятому узлу. Подобно муравьиной колонии, сеть биткойн является гибкой сетью простых узлов, соблюдающих простые правила, и эти узлы, объединенные в сеть, могут делать удивительные вещи без какой-либо централизованной координации.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения:

Курсив

Используется для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также имен и расширений файлов и каталогов.

Моноширинный шрифт
: Используется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

Моноширинный полужирный шрифт
: Используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

Моноширинный курсив
: Используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.


Такая пиктограмма обозначает совет или рекомендацию.


Такая пиктограмма обозначает указание или примечание общего характера.


Эта пиктограмма обозначает предупреждение или особое внимание к потенциально опасным объектам.

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 прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы 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.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку

в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Адреса биткойн-систем и транзакций в книге

Почти все адреса биткойн-систем, транзакций, ключей, QR-кодов и данных блокчейна, используемых в этой книге, являются реальными. Это означает, что вы можете просматривать структуры данных блокчейна, изучать транзакции, предлагаемые как примеры, использовать их в собственных скриптах и программах и т. д.

Но следует отметить, что секретные ключи, использованные для формирования адресов, либо опубликованы в этой книге, либо уже «засвечены» (таким образом, секретными уже не являются). И если вы отправите деньги на любой из этих адресов, то деньги будут безвозвратно потеряны или в некоторых случаях кто-то, прочитавший данную книгу, сможет присвоить их, воспользовавшись опубликованными здесь секретными ключами.

Не посылайте деньги по какому-либо адресу, приведенному в этой книге. Деньги будут присвоены другим читателем или исчезнут безвозвратно.

Как связаться с автором

С автором книги Андреасом М. Антонопулосом можно связаться через его личный сайт: https://antonopoulos.com/.

Информация о книге «Mastering Bitcoin», а также о платформе Open Edition и переводах книги на другие языки доступна на сайте: https://bitcoinbook.info/.

Автор в соцсети Facebook: https://facebook.com/AndreasMAntonopoulos.

Автор в Twitter: https://twitter.com/aantonop.

Автор в Linkedin: https://linkedin.com/company/aantonop.

Автор благодарит всех, кто поддерживает его работу ежемесячными безвозмездными взносами.

Страница автора на сайте Patreon: https://patreon.com/aantonop.

Благодарности

Эта книга представляет собой результат труда многих людей, внесших свой вклад. Я благодарен за всю помощь, которую я получил от друзей, коллег и даже совершенно незнакомых людей, подключившихся к моей работе над полноценной технической книгой о криптографических валютах и биткойне.

Невозможно отделить технологию биткойна от биткойн-сообщества, поэтому книга о технологии биткойна появилась во многом благодаря биткойн-сообществу, которое вдохновляло, поддерживало и поощряло мою работу от начала до конца. Эта книга, как ничто другое, позволила мне стать частью замечательного сообщества на два года, за что я безмерно ему благодарен. Очень трудно назвать по именам всех людей, с которыми я беседовал на конференциях, мероприятиях, семинарах, неформальных встречах, вечеринках и небольших частных собраниях, а также всех, кто общался со мной через Twitter, Reddit, на форуме bitcointalk.org и на GitHub, в общем, всех, кто так или иначе оказал влияние на эту книгу. Все идеи, аналогии, вопросы, ответы и объяснения, которые вы найдете в этой книге, были в той или иной степени предложены, проверены или улучшены с помощью сообщества. Спасибо всем за поддержку, без вас эта книга не появилась бы на свет. Я бесконечно благодарен вам.

Разумеется, путь к написанию книг начался гораздо раньше. Моим первым языком (в школе) был греческий, поэтому пришлось пройти коррективный курс английского письменного на первом курсе университета. Я выражаю благодарность Диане Кордас (Diana Kordas), моему преподавателю английского письменного, которая помогла мне обрести уверенность и прочные навыки в течение того года. Позже, уже как профессионал, я развил свои навыки технического писателя, публикуя статьи о центрах данных в журнале Network World. Благодарю Джона Дикса (John Dix) и Джона Галланта (Jon Gallant), предоставивших мне первую рабочую должность обозревателя-колумниста в Network World, редактора Майкла Куни (Michael Cooney) и коллегу Джона Тилл Джонсон (Johna Till Johnson), которые редактировали мои обзоры и готовили их к публикации. 500 слов в неделю в течение четырех лет дали мне достаточный опыт, и я окончательно решил заняться написанием книг.

Спасибо также тем, кто поддержал меня, когда я предложил свою книгу издательству O'Reilly. Отдельная благодарность Джону Галланту (John Gallant), Грегори Нессу (Gregory Ness), Ричарду Стиннену (Richard Stiennon), Джоелю Снайдеру (Joel Snyder), Эдаму Б. Ливайну (Adam B. Levine), Сандре Гиттлин (Sandra Gittlen), Джону Диксу (John Dix), Джоне Тилл Джонсон (Johna Till Johnson), Роджеру Веру (Roger Ver) и Йону Матонису (Jon Matonis). Особая благодарность Ричарду Кэгэну (Richard Kagan) и Таймону Маттошко (Tymon Mattoszko) за обзоры и рецензии ранних версий книги и Мэтью Тэйлору (Matthew Taylor) за редактуру и корректуру.

Спасибо Крикет Лью (Cricket Liu), автору книги DNS и BIND, который представил меня издательству O'Reilly. Также благодарю Майкла Лукидеса (Michael Loukides) и Элисон Макдоналд (Allyson MacDonald) из O'Reilly, которые в течение нескольких месяцев помогали моей книге появиться на свет. Элисон проявила особое терпение и такт, когда сроки выпуска книги оказывались под угрозой и повседневная жизнь с ее проблемами вмешивалась в издательские планы. За работу над вторым изданием благодарю Тимоти МакГоверна (Timothy McGovern) за общее руководство процессом, Ким Кофер (Kim Cofer) за внимательное и тщательное редактирование, а также Ребекку Панцер (Rebecca Panzer) за создание иллюстраций для множества новых схем.

Черновые наброски нескольких первых глав были самыми трудными, потому что биткойн – трудная тема сама по себе. Когда я начинал описывать один аспект технологии биткойна, неизбежно приходилось распутывать целый клубок взаимосвязанных аспектов. Приходилось многократно останавливать работу в слегка подавленном настроении, когда я безуспешно пытался упростить для понимания и доступно изложить такую обширную техническую тему. В конце концов, я решил рассказать историю биткойна с помощью рассказов людей, использующих технологию биткойна, и работать над книгой стало заметно легче. Я благодарю своего друга и наставника Ричарда Кэгэна (Richard Kagan), который помог распутать этот клубок проблем и преодолеть сложные моменты застоя. Спасибо Памеле Морган (Pamela Morgan), которая проверяла первые черновики каждой главы как в первом, так и во втором издании и задавала сложные вопросы, чтобы книга стала лучше. Также благодарю разработчиков из группы San Francisco Bitcoin Developers Meetup, Таарика Льюиса (Taariq Lewis) и Дениз Терри (Denise Terry) за помощь в проверке первого чернового материала. Спасибо Эндрю Ноглеру (Andrew Naugler) за дизайн инфографики.

Во время написания книги я открыл доступ к первым черновикам на сервисе GitHub и предложил всем желающим их прокомментировать. В ответ было получено более ста замечаний, предложений, исправлений и дополнений. Благодарю всех откликнувшихся на мое предложение, а их полный список можно посмотреть ниже, в разделе «Первый черновик (вклад сообщества на GitHub)». Особая благодарность – добровольным редакторам на GitHub Мин Т. Нгуену (Minh T. Nguyen) (1-е издание) и Уиллу Биннсу (Will Binns) (2-е издание), которые без устали регулировали, управляли и обрабатывали предложения, сообщения об ошибках и неточностях, а также исправляли ошибки непосредственно на GitHub.

После создания чернового варианта книги она прошла через несколько этапов технического редактирования и рецензирования. Спасибо Крикет Лью (Cricket Liu) и Лорне Ланц (Lorne Lantz) за тщательное рецензирование, комментарии и поддержку.

Несколько разработчиков, использующих технологию биткойна, прислало примеры кода, отзывы, комментарии и всячески поддерживало мою работу. Спасибо Амиру Тааки (Amir Taaki) и Эрику Воскуилу (Eric Voskuil) за предо-

ставленные фрагменты кода для примеров и множество полезных замечаний, Крису Клеешульте (Chris Kleeschulte) за материал, включенный в приложение по Bitcore, Виталику Бутерину (Vitalik Buterin) и Ричарду Киссу (Richard Kiss) за помощь с математикой эллиптических кривых и предоставление фрагментов кода, Гэвину Андресену (Gavin Andresen) за исправления, комментарии и поддержку, Михалису Каргакису (Michalis Kargakis) за комментарии, предложения и описание btcd, Робину Инге (Robin Inge) за поиск опечаток и ошибок, что несомненно улучшило печатное издание. При работе над вторым изданием я снова получил огромную помощь от многих разработчиков Bitcoin Core, в том числе от Эрика Ломброзо (Eric Lombrozo), открывшего тайны Segregated Witness, от Люка-младшего (Luke-Jr), который помог улучшить главу о транзакциях, от Джонсона Лау (Johnson Lau), рецензировавшего Segregated Witness и другие главы, и от многих других. Благодарю Джозефа Пуна (Joseph Poon), Тадже Драйа (Tadge Dryja) и Олаолува Осантокана (Olaoluwa Osuntokun), которые предоставили описание сети Lightning Network, рецензировали мои материалы, отвечали на вопросы, которые вызывали у меня затруднения.

Своей любовью к печатному слову и к книгам я обязан моей матери Терезе, вырастившей меня в доме, в котором книжные полки и шкафы стояли буквально у каждой стены. Моя мама купила мне самый первый компьютер в 1982 году, хотя сама считала себя технофобом. Мой отец, Менелаос, инженер-строитель, который недавно опубликовал свою первую книгу в возрасте 80 лет, научил меня логическому и аналитическому мышлению и любви к науке и технике.

Спасибо всем за поддержку на протяжении всего этого пути.

Первый черновик (вклад сообщества на GitHub)

Многие люди внесли свой вклад, предлагая комментарии, исправления и дополнения в самую первую черновую версию книги на GitHub. Благодарю всех за участие в создании этой книги.

Ниже приведен список самых активных участников процесса подготовки первой версии книги на GitHub с указанием в скобках идентификаторов их учетных записей:

- Алекс Уотерс (Alex Waters, alexwaters);
- Эндрю Доналд Кеннеди (Andrew Donald Kennedy, grkvlt);
- bitcoinctf;
- Брайан Гмырек (Bryan Gmyrek, physicsdude);
- Кейси Флинн (Casey Flynn, cflynn07);
- Чэпмэн Шуп (Chapman Shoop, belovachap);
- Кристи Д'Анна (Christie D'Anna, avocadobreath);
- Коди Скотт (Cody Scott, Siecje);
- coinradar;
- Крэджин Годли (Cragin Godley, cgodley);
- dallyshalla;

- Диего Виола (Diego Viola, diegoviola);
- Дирк Якель (Dirk Jäckel, biafra23);
- Димитрис Цапакидис (Dimitris Tsapakidis, dimitris-t);
- Дмитрий Маракасов (Dmitry Marakasov, AMDmi3);
- drstrangeM;
- Эд Айкхолт (Ed Eykholt, edeykholt);
- Эд Лиф (Ed Leafe, EdLeafe);
- Эдвард Поснак (Edward Posnak, edposnak);
- Элиас Родригес (Elias Rodrigues, elias19r);
- Эрик Воскуил (Eric Voskuil, evoskuil);
- Эрик Уинчелл (Eric Winchell, winchell);
- Эрик Вальстрём (Erik Wahlström, erikwam);
- effectsToCause (vericoin);
- Эстебан Ордано (Esteban Ordano, eordano);
- ethers;
- fabienhinault;
- Франк Хёгер (Frank Höger, francyi);
- Гаурав Рана (Gaurav Rana, bitcoinsSG);
- genjix;
- halseth;
- Хольгер Шинцель (Holger Schinzel, schinzelh);
- Иоаннис Керувим (Ioannis Cherouvim, cherouvim);
- Айш От, младший (Ish Ot Jr., ishotjr);
- Джеймс Эддисон (James Addison, jayaddison);
- Джеймсон Лопп (Jameson Lopp, jlopp);
- Джейсон Бистерфельдт (Jason Bisterfeldt, jbisterfeldt);
- Хавьер Рохас (Javier Rojas, fjrojasgarcia);
- Джереми Бокобца (Jeremy Bokobza, bokobza);
- JerJohn15;
- Джо Бауэрс (Joe Bauers, joebauers);
- joflynn;
- Джонсон Лау (Johnson Lau, jl2012);
- Джонатан Кросс (Jonathan Cross, jonathancross);
- Jorgeminator;
- Кай Баккер (Kai Bakker, kaibakker);
- Май-Суан Чиа (Mai-Hsuan Chia, mhchia);
- Marzig (marzig76);
- Максимилиан Райхель (Maximilian Reichel, phramz);
- Михалис Каргакис (Michalis Kargakis, kargakis);
- Микаэль С. Ипполито (Michael C. Ippolito, michaelcippolito);
- Михаил Руссу (Mihail Russu, MihailRussu);
- Мин Т. Нгуен (Minh T. Nguyen, enderminh);
- Нагарай Хубли (Nagaraj Hubli, nagarajhubli);

- Nekomata (nekomata-3);
- Роберт Ферс (Robert Furse, Rfurse);
- Ричард Кисс (Richard Kiss, richardkiss);
- Рубен Александер (Ruben Alexander, hizzvizz);
- Сэм Ричи (Sam Ritchie, sritchie);
- Сергей Котляр (Sergej Kotliar, ziggamon);
- Сейичи Учида (Seiichi Uchida, topecongiro);
- Симон де ла Рувьер (Simon de la Rouviere, simondlr);
- Стефан Усте (Stephan Oeste, Emzy);
- takaya-imai;
- Тьяго Арраис (Thiago Arrais, thiagoarrais);
- venzen;
- Уилл Биннс (Will Binns, wbnns);
- wintercooled;
- wjx;
- Войцех Лангиевич (Wojciech Langiewicz, wlk);
- yurigeorgiev4.

Глава 1

Введение

Что такое биткойн

Биткойн (bitcoin) – это набор концепций и технологий, которые формируют основу цифровой денежной экосистемы. Денежные единицы, называемые биткойнами, используются для хранения и передачи ценности в денежном выражении между членами биткойн-сети. Пользователи биткойн-системы обмениваются информацией друг с другом, используя для этого протокол биткойна, работающий в основном через Интернет, хотя могут применяться и любые другие транспортные сетевые протоколы. Стек протоколов биткойна, доступный в виде ПО с открытыми исходными кодами, может быть реализован на многочисленных типах устройств, в том числе на ноутбуках и смартфонах, что существенно увеличивает массовую доступность этой технологии.

Пользователи могут передавать биткойны по сети, чтобы выполнять с ними практически те же операции, что с традиционными денежными средствами, в том числе покупать и продавать товары, пересылать деньги людям и организациям или предоставлять кредит. Биткойны можно покупать, продавать и обменивать на другие валюты на специализированных валютных биржах. В некотором смысле биткойн является идеальной формой денег для Интернета благодаря скорости операций с ним, защищенности и безграничности области его применения.

В отличие от обычных денежных единиц, биткойн абсолютно виртуален. Не существует ни физических денежных знаков, ни даже цифровых денежных знаков для биткойна. Воображаемые денежные единицы участвуют в транзакциях, которые передают какие-либо ценности (в стоимостном выражении) от отправителя к получателю. Пользователи биткойна владеют ключами, которые позволяют им подтверждать обладание биткойнами в биткойн-сети. С помощью этих ключей пользователи могут подписывать (заверять) транзакции для получения доступа к своей валюте и ее расходования посредством передачи новому владельцу. Ключи часто хранятся в цифровом кошельке на компьютере или смартфоне каждого пользователя. Обладание ключом, с помощью которого можно заверить транзакцию, является единственным предварительно

ным условием для операций с биткойнами, при этом управление полностью передается каждому пользователю.

Биткойн представляет собой распределенную пиринговую (peer-to-peer) (или одноранговую) систему. Это означает, что в ней нет «центрального» сервера или какого-либо пункта управления. Биткойны создаются с помощью процесса, называемого майнингом (mining), который подразумевает конкуренцию в поиске решений для математической задачи при обработке транзакций биткойнов. Любой член биткойн-сети (то есть любой, использующий устройство, на котором работает полный стек протоколов биткойна) может выступить в роли майнера (miner), используя вычислительные мощности своего компьютера для проверки и фиксации транзакций. В среднем через каждые 10 минут кто-то побеждает в состязании за право подтверждения корректности транзакций, выполненных за эти прошедшие 10 минут, и вознаграждается за это новым биткойном. По существу, майнинг биткойнов способствует децентрализации функций клиринга и выпуска денежных знаков центральным банком и фактически исключает необходимость в каком-либо центральном банке.

Протокол биткойна включает встроенные алгоритмы, которые управляют функцией майнинга в сетевой среде. Сложность вычислительной задачи, которую обязательно должны выполнить майнеры, регулируется динамически, поэтому в среднем через каждые 10 минут кто-то достигает успеха вне зависимости от количества майнеров (и от количества обрабатываемых задач), конкурирующих в текущий момент. Кроме того, протокол предусматривает уменьшение наполовину скорости создания новых биткойнов через каждые 4 года и ограничивает общее количество созданных биткойнов фиксированной величиной, которая не должна превышать сумму в 21 миллион единиц. Таким образом, количество биткойнов, находящихся в обращении, весьма точно описывается легко прогнозируемой кривой, которая достигнет значения 21 миллион к 2140 году. Благодаря такому уменьшению скорости «эмиссии» в течение длительного интервала времени биткойн представляет собой дефляционную валюту. Более того, биткойн не подвержен инфляции в форме «печатания» новых денежных купюр сверх предполагаемой эмиссионной нормы.

Если заглянуть поглубже, то биткойн также можно определить как название протокола, пиринговой сети и новой технологии распределенной обработки данных. Биткойн как денежная единица действительно представляет собой самое первое практическое приложение этой новой технологии. Биткойн является суммарным результатом многолетних исследований в области криптографии и распределенных систем и включает четыре главные инновации, объединенные в единственную в своем роде мощную комбинацию. Основными компонентами технологии биткойна являются:

- децентрализованная пиринговая сеть (протокол биткойна);
- общедоступный реестр транзакций (блокчейн (blockchain));
- набор правил для независимой проверки (валидации) транзакций и эмиссии (выпуска) денежных единиц (правила консенсуса);

- механизм для достижения глобального децентрализованного (распределенного) консенсуса при проверке корректности (валидации) блокчейна (алгоритм доказательства выполнения работы, Proof-of-Work algorithm).

Как разработчик я считаю биткойн системой, очень похожей на «Интернет денег» (Internet of money), то есть на сеть для распространения ценностей и для защиты права владения цифровыми активами, функционирующую на основе распределенных вычислений. При этом роль биткойна гораздо более значимая, чем кажется на первый взгляд.

В этой главе мы начнем изучение некоторых основных концепций и определений, познакомимся с необходимым программным обеспечением (ПО) и попробуем использовать биткойн для простых транзакций. В следующих главах будут рассматриваться более глубокие уровни технологии, которая сделала возможным появление биткойна, а также внутренние функциональные возможности и особенности сети и протокола биткойна.

Цифровые деньги до биткойна

Появление жизнеспособных цифровых денег тесно связано с разработками в области криптографии. Поэтому вполне естественно считать основополагающей главной задачей использование битов для представления стоимостной ценности, которую можно обменивать на товары и услуги. Все желающие пользоваться цифровыми деньгами должны ответить на три ключевых вопроса:

1. Могу ли я быть уверенным в том, что цифровые деньги достоверны (и законны) и не являются поддельными (фальшивыми)?
2. Могу ли я быть уверенным в том, что цифровые деньги можно потратить только один раз (эта проблема известна под названием «двойное расходование» (double spending))?
3. Могу ли я быть уверенным в том, что никто другой не сможет заявить, что эти деньги принадлежат ему, а не мне?

Организации, ведающие выпуском бумажных денег, ведут непрерывную борьбу с проблемой подделки купюр, используя для этого всё более сложные степени защиты бумаги и технологии печати. Бумажные деньги решают проблему двойного расходования очень просто, потому что один и тот же лист бумаги не может находиться в двух местах одновременно. Разумеется, и привычные всем нам деньги часто можно хранить и передавать в цифровой форме. В этих случаях проблемы подделки и двойного расходования устраняются с помощью процедуры клиринга (безналичных взаимных расчетов) всех электронных транзакций в централизованной системе компетентных органов, осуществляющих общий мониторинг всех денежных средств, находящихся в обращении. Для цифровых денег, которые не могут воспользоваться преимуществами невидимой типографской краски или голографических полосок, криптография предлагает основное средство для создания уверенности в законности объявляемой пользователем ценности. Точнее говоря, криптографические цифровые подписи позволяют пользователю заверить (собственной подписью) цифровые активы или транзакцию, подтверждающую

право владения этим активом. При использовании соответствующей архитектуры цифровые подписи также могут применяться для устранения проблемы двойного расходования.

С началом более широкой доступности и более глубокого понимания методов криптографии в конце 1980-х гг. многие исследователи попытались использовать криптографию для создания цифровых валют. Самые первые проекты в этой области генерировали цифровые деньги, обычно обеспечиваемые национальной валютой или драгоценными металлами, например золотом.

Несмотря на то что эти первые цифровые деньги действительно функционировали, они оставались в рамках централизованных систем, поэтому представляли собой легкую мишень для нападений государственных органов и хакеров. Первые цифровые деньги использовали централизованную расчетную палату для регулирования выполнения всех транзакций с регулярными интервалами точно так же, как в обычной банковской системе. К сожалению, в большинстве случаев эти находящиеся в ранней стадии становления цифровые деньги привлекали особое внимание встревоженных правительственных органов и в конечном итоге уничтожались в судебном порядке. Иногда крах цифровых валют становился заметным явлением, когда поддерживающие их компании внезапно ликвидировались. Чтобы стать устойчивыми к воздействиям противников, будь то официальные правительственные органы или криминальные элементы, децентрализованные цифровые деньги должны были непременно избавиться от единственной уязвимой для атак точки. Биткойн является именно такой системой, децентрализованной по своей сущности изначально и свободной от каких-либо центральных органов авторизации или пунктов управления, которые можно атаковать и вывести из строя.

История создания биткойна

Биткойн был создан в 2008 году, о чем сообщала статья «Bitcoin: A Peer-to-Peer Electronic Cash System»[1], опубликованная под псевдонимом Сатоши Накамото (Satoshi Nakamoto) (см. приложение А). Накамото объединил несколько более ранних инноваций, таких как b-money и HashCash, для создания полностью децентрализованной электронной системы денежных расчётов, в основе которой не имелось какого бы то ни было центрального органа авторизации для эмиссии денежных единиц или для регулирования и проверки корректности транзакций. Главным нововведением стало использование распределенной системы вычислений (названной алгоритмом доказательства выполнения работы, Proof-of-Work algorithm) для проведения всеобщих «выборов» через каждые 10 минут, что позволяло в децентрализованной сети достигать консенсуса (consensus) по текущему состоянию транзакций. Такой подход изящно решил

[1] «Bitcoin: A Peer-to-Peer Electronic Cash System», Satoshi Nakamoto (https://bitcoin.org/bitcoin.pdf).

проблему двойного расходования, при возникновении которой одна денежная единица может быть потрачена дважды. До этого проблема двойного расходования была явным недостатком цифровых денег и решалась операциями клиринга всех транзакций через центральную расчётную палату.

Биткойн-сеть начала свою работу в 2009 году на основе реализации, описанной в статье Накамото и с тех пор многократно улучшенной многими другими программистами. Реализация алгоритма доказательства выполнения работы (майнинга), обеспечивающего защиту и жизнеспособность биткойна, постоянно наращивала свою мощь и в настоящее время превосходит суммарную вычислительную мощность самых лучших суперкомпьютеров мира. Общая рыночная стоимость биткойна временами превышает сумму в 20 миллиардов долларов США в зависимости от текущего курса обмена биткойна на доллар. До сего момента самой крупной транзакцией, проведенной в биткойн-сети, была сумма в 150 миллионов долларов США, переведенная мгновенно и обработанная без каких-либо отчислений.

Сатоши Накамото отстранился от активной деятельности в апреле 2011 года и передал ответственность за разработку программного кода и развитие сети преуспевающей группе добровольцев. Настоящее имя человека или группы людей, придумавших биткойн, остается неизвестным. В любом случае, ни Сатоши Накамото, ни кто-либо другой не пытался лично управлять всей биткойн-системой в целом. Функциональность биткойн-системы основана на совершенно ясных математических принципах, на открытых исходных кодах и на консенсусе (согласовании) между членами системы. Само по себе это изобретение стало прорывом и уже породило новую область науки на стыке таких дисциплин, как распределенная обработка данных, экономика и эконометрика (математическая экономика).

Решение проблемы распределенной обработки данных

Изобретение Сатоши Накамото, кроме всего прочего, представляет собой практическое и совершенно новое решение одной из задач распределенной обработки данных, известной под названием «Задача византийских генералов». Краткое объяснение задачи: попытаться согласовать образ действий или состояние системы посредством обмена информацией в ненадежной и потенциально опасной сетевой среде. Решение Сатоши Накамото, использующее концепцию доказательства выполнения работы для достижения консенсуса без центрального органа управления, заслуживающего доверия, является настоящим прорывом в области распределенной обработки данных, а область применения этого решения не ограничивается финансовой сферой. Эту технологию можно применять для достижения консенсуса в децентрализованных сетях для доказательства честности и корректности процедур голосования при выборах, результатов тиражей лотерей, реестров имущества (активов, фондов и т. п.), цифровых нотариальных свидетельств и многого другого.

Варианты использования биткойнов, пользователи и их истории

Биткойн – это инновация в древней технологии денежного оборота. По своей сути деньги просто обеспечивают обмен разнообразными ценностями между людьми. Поэтому, чтобы полностью понять биткойн и его практическое применение, мы начнем знакомство с этой технологией с точки зрения людей, использующих ее. Каждый из упомянутых ниже людей и соответствующих рассказов о них показывает один или несколько конкретных вариантов использования. Мы будем возвращаться к ним на протяжении всей книги:

Розничная торговля дешевыми товарами в Северной Америке

Алиса (Alice) живет в области залива в Северной Калифорнии. Она услышала о биткойне от своих друзей-технарей и хочет использовать его. Мы будем следить за тем, как Алиса изучает биткойн, зарабатывает несколько биткойнов, затем тратит несколько биткойнов на чашку кофе в кафе Боба в Пало Алто. Эта история познакомит нас с программным обеспечением, с процедурами обмена и с основными простыми транзакциями с точки зрения потребителя розничных товаров.

Розничная торговля дорогостоящими товарами в Северной Америке

Кэрол (Carol) – владелица художественной галереи в Сан-Франциско. Она продает дорогостоящие картины за биткойны. В этой истории будет описана опасность атак типа 51 процент при достижении консенсуса для розничных продавцов дорогостоящих товаров.

Услуги по офшорному контракту

Боб (Bob), владелец кафе в Пало Алто, создает новый веб-сайт. Он заключил контракт с веб-разработчиком Гопешем (Gopesh), который живет в Бангалоре (Индия). Гопеш согласен получить оплату в биткойнах. Эта история демонстрирует возможности биткойна в области аутсорсинга, при получении услуг по контракту и при выполнении международных денежных переводов.

Веб-магазин

Габриэль (Gabriel) – предприимчивый юноша из Рио-де-Жанейро, организовал небольшой веб-магазин, в котором продаются футболки, кофейные кружки и наклейки с логотипом биткойна. Габриэль слишком молод, чтобы открыть личный счет в банке, но родители всячески поощряют его тягу к предпринимательству.

Благотворительная деятельность, пожертвования

Эухения (Eugenia) – директор детского благотворительного учреждения на Филиппинах. Недавно она узнала при биткойн и хочет воспользоваться им для охвата новой крупной группы зарубежных и местных жертвователей,

чтобы организовать сбор средств для своей благотворительной деятельности. Кроме того, она изучает способы применения биткойна для быстрой передачи денежных средств нуждающимся. Эта история продемонстрирует использование биткойна для организации сбора денежных средств без учета различий валют и государственных границ, а также использование общедоступного реестра для честного ведения дел в благотворительных организациях.

Импорт/экспорт
Мохаммед (Mohammed) – импортер электроники в Дубаи. Он пытается использовать биткойн для покупки электроники в США и Китае для импорта в ОАЭ, чтобы ускорить процедуру оплаты импортируемых товаров. Эта история покажет, как можно использовать биткойн для крупных международных бизнес-платежей, связанных с материальными товарами.

Майнинг биткойнов
Цзин (Jing) – студент, изучающий компьютерную инженерию в Шанхае. Он собрал стойку с блоками майнинга для зарабатывания биткойнов, используя свои инженерные навыки и знания для обеспечения прибыли. В этой истории будет рассматриваться «промышленная» основа биткойна: специализированное оборудование, применяемое для защиты биткойн-сети и для генерации новых денежных единиц.

В каждой из этих историй фигурируют реальные люди, предприятия и организации, использующие в настоящее время биткойн для создания новых рынков, новых отраслей промышленности и современных эффективных решений проблем глобальной экономики.

Начинаем обучение

Биткойн (bitcoin) – это протокол, доступ к которому можно получить с помощью клиентского приложения, говорящего на языке этого протокола. Биткойн-кошелек (bitcoin wallet) представляет собой наиболее часто используемый пользовательский интерфейс к биткойн-системе, точно так же, как веб-браузер является наиболее часто используемым пользовательским интерфейсом к протоколу HTTP. Существует множество реализаций и вариантов биткойн-кошельков, подобно множеству вариантов веб-браузеров (например, Chrome, Safari, Firefox, Internet Explorer, Яндекс-браузер). У каждого есть свой любимый браузер (я за Mozilla Firefox) и свой «отрицательный герой» (я против Internet Explorer), так и биткойн-кошельки различаются по качеству, производительности, защищенности, уровню секретности (приватности) и надежности. Существует также эталонная реализация протокола биткойна, известная как Satoshi Client или Bitcoin Core, производная от исходной реализации, написанной Сатоши Накамото.

Выбор биткойн-кошелька

Биткойн-кошельки являются наиболее активно разрабатываемыми приложениями в экосистеме биткойна. Здесь существует жесткая конкуренция, и, несмотря на то что очередной новый кошелек, возможно, разрабатывается прямо сейчас, несколько кошельков, появившихся в прошлом году, уже не имеет активной поддержки. Многие кошельки предназначены для конкретных платформ или для специальных вариантов использования, некоторые больше подходят для начинающих, тогда как другие предоставляют полный набор функциональных возможностей для более опытных пользователей. Выбор кошелька абсолютно индивидуален и зависит от способа использования и практического опыта пользователя. Таким образом, невозможно порекомендовать конкретное название или конкретный проект кошелька. Тем не менее можно провести классификацию биткойн-кошельков в соответствии с поддерживаемыми платформами и функциональными возможностями и внести определенную ясность в информацию о различных типах существующих кошельков. Хорошо, что перемещение денег между кошельками производится просто, дешево и быстро, поэтому лучше всего попробовать несколько разных кошельков, чтобы выбрать тот, который больше всего соответствует вашим требованиям.

По поддерживаемым платформам можно классифицировать биткойн-кошельки следующим образом:

- *кошелек для десктопа (desktop wallet)* – первый тип биткойн-кошелька, созданный как эталонная реализация, и многие пользователи работают с кошельками для десктопа (настольного компьютера), выбирая их за функциональные возможности, автономность и полноту управления, которые предлагает этот тип кошельков. Но работа под управлением операционных систем общего назначения, таких как Windows и MacOS, имеет свои недостатки, связанные с недостаточной защищенностью, так что десктоп-платформы зачастую не могут обеспечить надлежащий уровень защиты и соответствующую конфигурацию;
- *мобильный кошелек (mobile wallet)* – наиболее часто используемый тип биткойн-кошелька. Работающие под управлением операционных систем для смартфонов, таких как Apple iOS и Android, эти кошельки чаще всего являются наилучшим выбором для новых пользователей. Многие из них спроектированы таким образом, чтобы обеспечить максимальную простоту использования, но есть и полнофункциональные мобильные кошельки для опытных пользователей;
- *веб-кошелек (web wallet)* – доступен через веб-браузер, а сам пользовательский кошелек хранится на сервере, принадлежащем третьей стороне. Это похоже на организацию веб-почты, полностью основанной на использовании стороннего сервера. Некоторые из этих сервисов используют для работы код на стороне клиента, выполняемый в браузере пользователя, что позволяет сохранить управление ключами биткойна

в руках пользователя. Но большинство сервисов предлагает компромисс, принимая на себя управление ключами биткойна в обмен на простоту использования. И всё же я не рекомендую хранить большой объем биткойнов на сторонних системах;

- *аппаратный кошелек (hardware wallet)* – устройство, обеспечивающее защиту биткойн-кошелька, хранящегося на специализированных аппаратных средствах. Такое устройство может работать вместе с десктопным веб-браузером, обмениваясь данными через порт USB или через ближнюю бесконтактную связь (near-field-communication (NFC)) на мобильном устройстве. Благодаря обработке всех операций с биткойнами на специализированной аппаратуре этот тип кошельков считается очень хорошо защищенным и вполне подходящим для хранения крупных сумм в биткойнах;
- *бумажный кошелек (paper wallet)* – ключи, управляющие биткойном, также могут быть распечатаны для долговременного хранения. Это называют бумажным кошельком, несмотря на то что печать может производиться и на других материалах (дерево, металл и т. д.). Бумажные кошельки предлагают низкотехнологичные, но весьма защищенные средства хранения биткойнов в течение длительного времени. Офлайн-хранилище также часто называют «холодильным хранением» (cold storage).

Другой способ классификации биткойн-кошельков – по степени их независимости (возможности автономного функционирования) и по способу их взаимодействия с биткойн-сетью:

- *полноценный клиент (full client)*, или *«полноценный узел» (full node)*, – это клиент, который хранит полную хронологию транзакций биткойнов (каждую транзакцию, когда-либо выполненную любым пользователем), управляет кошельками пользователей и может непосредственно начать выполнение транзакций в биткойн-сети. Полноценный клиент имеет дело со всеми аспектами протокола и способен независимо проверять корректность всей структуры блокчейна и любой транзакции. Полноценный клиент потребляет довольно-таки существенные ресурсы компьютера (например, более 125 Гб дисковой памяти, 2 Гб оперативной памяти), но при этом обеспечивает полную автономию и независимую верификацию транзакций;
- *упрощенный клиент (lightweight client)* – также известен под названием «клиент с упрощенной проверкой платежей» (simple-payment-verification (SPV) client), соединяется с полноценным узлом биткойн-сети (описанным выше) для доступа к информации о транзакциях биткойнов, но хранит пользовательский кошелек локально и независимо создает, проверяет и пересылает транзакции. Упрощенные клиенты взаимодействуют с биткойн-сетью напрямую, без каких-либо посредников;

- *клиент с прикладным программным интерфейсом (API) стороннего производителя (third-party API client)* – взаимодействует с биткойн-сетью через систему прикладных программных интерфейсов (API) стороннего производителя (третьей стороны) вместо прямого соединения с биткойн-сетью. Кошелек может храниться у самого пользователя или на сторонних серверах, но все транзакции выполняются через посредника (третью сторону).

Многие биткойн-кошельки попадают сразу в несколько групп классификации, поскольку в них объединены различные классификационные характеристики, но наиболее часто встречаются три комбинации: настольный (десктоп) полноценный клиент, мобильный упрощенный кошелек и веб-кошелек с прикладным программным интерфейсом от стороннего производителя. Чаще всего границы между описанными выше категориями слабо различимы, так как многие кошельки работают на нескольких платформах и могут взаимодействовать с сетью разнообразными способами.

В этой книге мы рассмотрим использование разнообразных загружаемых биткойн-клиентов, от эталонной реализации (Bitcoin Core) до мобильных и веб-кошельков. В некоторых примерах потребуется применение эталонной реализации Bitcoin Core, которая, будучи полноценным клиентом, кроме того, предоставляет прикладные программные интерфейсы (API) к кошельку, сети и сервисам выполнения транзакций. Если вы намерены глубже исследовать программные интерфейсы в биткойн-системе, то вам необходима работающая эталонная реализация Bitcoin Core или один из альтернативных клиентов (см. раздел «Альтернативные клиенты, библиотеки и инструментальные пакеты разработчика» в главе 3).

Сразу переходим к делу

Алиса, с которой мы познакомились немного раньше, в разделе «Варианты использования биткойнов, пользователи и их истории», – пользователь с минимумом технических знаний. Она совсем недавно узнала о биткойне от своего друга Джо (Joe). Как-то на вечеринке Джо в очередной раз с энтузиазмом рассказывал о биткойне всем присутствующим и даже предлагал продемонстрировать работу с ним. Заинтригованная его рассказом, Алиса спросила, с чего начать использование биткойна. Джо ответил, что для новичков лучшим выбором будет мобильный кошелек, и порекомендовал несколько предпочтительных, с его точки зрения, вариантов. Алиса загрузила программу Mycelium для платформы Android и установила ее на своем мобильнике.

Когда Алиса впервые запускает Mycelium, то, как и большинство биткойн-кошельков, это приложение автоматически создает новый кошелек для нового пользователя. Алиса видит на экране этот кошелек, выглядящий так, как показано на рис. 1.1 (примечание: не посылайте биткойны на адрес из этого примера, вы потеряете их навсегда).

Рис. 1.1 ❖ Мобильный кошелек Mycelium

Самая важная часть изображения на показанном экране – биткойн-адрес (bitcoin address) Алисы, который выглядит как длинная строка букв и цифр: `1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK`. Справа от биткойн-адреса кошелька расположен QR-код, особая форма штрихового кода, содержащая ту же информацию в формате, который может быть отсканирован видеокамерой смартфона. QR-код представляет собой квадрат с рисунком, составленным из черных и белых точек. Алиса может скопировать биткойн-адрес или QR-код в буфер обмена простым касанием (tapping) изображения QR-кода или кнопки **Receive** (Принять). В большинстве кошельков касание QR-кода также увеличивает его изображение, чтобы упростить его сканирование камерой смартфона.

Биткойн-адреса начинаются с цифр 1 или 3. Как и адреса электронной почты, биткойн-адреса можно сообщать другим пользователям биткойн-системы, которые воспользуются ими для пересылки биткойнов непосредственно в ваш кошелек. С точки зрения безопасности распространение своего биткойн-адреса не связано с каким-либо риском. Биткойн-адрес можно публиковать везде без угрозы для своей учетной записи. В отличие от адресов электронной почты, новые биткойн-адреса можно создавать без ограничений, и все они будут направлять денежные средства в ваш кошелек. В действительности многие новейшие версии кошельков автоматически создают новый адрес для каждой очередной транзакции в целях обеспечения максимальной секретности. Кошелек – это просто набор адресов и ключей, открывающих доступ к хранящимся в нем денежным средствам.

Теперь Алиса готова к получению цифровых денег. Прикладная программа кошелька случайным образом сгенерировала секретный ключ (private key) (более подробно секретный ключ будет описан в разделе «Секретные ключи» главы 4) вместе с соответствующим ему биткойн-адресом. В этот момент ее биткойн-адрес неизвестен в биткойн-сети и не «зарегистрирован» в какой-либо части биткойн-системы. Этот биткойн-адрес – пока просто число, соответствующее ключу, который Алиса может использовать для управления доступом к своим денежным средствам. Биткойн-адрес сгенерирован кошельком автономно без обращения или регистрации на каком-либо сервисе. В действительности большинство кошельков не устанавливает никакой связи между биткойн-адресом и какой-либо внешней идентифицируемой информацией, включая данные, подтверждающие личность пользователя. До того момента, когда этот адрес определяется как получатель денежных единиц в результате транзакций, зафиксированных в реестре биткойна, он является просто частью огромного набора возможных адресов, допустимых в биткойн-системе. Только после связывания такого адреса с некоторой транзакцией он становится частью списка известных адресов в биткойн-сети.

Итак, Алиса готова начать практическое использование своего нового биткойн-кошелька.

Получаем свой первый биткойн

Первой и зачастую самой сложной задачей для новых пользователей является получение хотя бы нескольких биткойнов. В отличие от иностранных валют, невозможно купить биткойны в банке или в пункте обмена валют.

Биткойн-транзакции необратимы. Трансферы в большинстве электронных платежных сетевых систем, таких как кредитные карты, дебетовые карты, PayPal и банковские счета, являются обратимыми. Для любого продавца биткойнов это различие создает вполне реальную опасность, состоящую в том, что покупатель может отозвать свой электронный платеж после того, как получит биткойны, то есть, в сущности, обманет продавца. Чтобы как-то уменьшить риск, компании, принимающие обычные электронные платежи в обмен на биткойны, обычно требуют от покупателей пройти процедуры проверки подлинности личности и подтверждения платежеспособности, которые могут продолжаться от нескольких дней до нескольких недель. Это означает, что как новый пользователь вы не имеете возможности мгновенно купить биткойны с помощью кредитной карты. Но немного терпения и творческого мышления – и такой способ вам не понадобится.

Ниже описаны некоторые методы получения биткойнов новыми пользователями:

- найдите друга (знакомого), у которого есть биткойны, и купите у него несколько единиц. Многие пользователи биткойн-систем начинали именно таким способом, потому что он наименее сложный. Найти лю-

дей с биткойнами можно на неформальной встрече местной группы биткойн-пользователей, о которых сообщается на сайте Meetup.com;

- воспользуйтесь надежным сервисом, например localbitcoins.com, чтобы найти продавца в вашем регионе и купить биткойны за наличные с оплатой при личной встрече;
- заработайте биткойны, продавая продукцию или услуги. Если вы программист, продавайте свои навыки и умения. Если вы парикмахер, стригите людей за биткойны;
- воспользуйтесь биткойн-банкоматом в вашем городе. Биткойн-банкомат – это устройство, которое принимает наличные деньги и пересылает биткойны в ваш биткойн-кошелек на смартфоне. Найти ближайший биткойн-банкомат можно с помощью онлайновой карты на сайте Coin ATM Radar (http://coinatmradar.com);
- воспользуйтесь обменным пунктом биткойнов, связанным с вашим банковским счетом. Сейчас во многих странах существуют обменные пункты, поддерживающие рынок покупателей и продавцов, обменивающих биткойны на местную валюту. В списках обменных курсов, например BitcoinAverage (https://bitcoinaverage.com), часто указываются обменные пункты, обменивающие биткойны на конкретную валюту.

Одним из преимуществ биткойна перед другими платежными системами, если использовать его правильно, является то, что биткойн обеспечивает гораздо бóльшую секретность для пользователей. Приобретение, хранение и расходование биткойнов не требуют разглашения личной информации и своих идентификационных данных или передачи таких данных кому бы то ни было. Но там, где биткойн имеет дело с обычными системами, например с валютными биржами или обменными пунктами, часто применяются государственные или международные законодательные нормы. Для обмена биткойнов на денежные единицы вашей страны от вас наверняка потребуют подтвердить свою личность и доказать подлинность банковской информации. Пользователи должны знать о том, что после одной операции с биткойнами, при которой была идентифицирована их личность, все последующие транзакции биткойнов будут так же легко идентифицироваться и прослеживаться. Это одна из причин, по которой многие пользователи предпочитают иметь учетные записи, предназначенные специально для обменных операций и не связанные напрямую с их кошельками.

Алисе рассказал о биткойнах ее друг, поэтому у нее есть возможность без затруднений приобрести свой первый биткойн. Далее мы увидим, как она покупает биткойн у своего друга Джо и как Джо пересылает этот биткойн в кошелек Алисы.

Поиск информации о текущей стоимости биткойна

Прежде чем Алиса сможет купить биткойн у Джо, они должны договориться об обменном курсе (exchange rate) между биткойном и долларами США. При этом у всех новичков возникает вполне естественный вопрос: «Кто устанавливает цену биткойна?» Короткий ответ: цена устанавливается рынком.

Биткойн, как и большинство других валют, имеет плавающий обменный курс (floating exchange rate). Это означает, что цена биткойна по отношению к любой другой валюте постоянно изменяется в зависимости от спроса и предложения на различных рынках, где имеет хождение биткойн. Например, «цена» биткойна в долларах США вычисляется отдельно на каждом рынке на основе самых последних операций по обмену биткойнов на доллары США. Таким образом, цена может изменяться несколько раз в секунду, и эти изменения следуют непрерывно. Информационные валютные сервисы объединяют данные о ценах с нескольких рынков и вычисляют средневзвешенное (с учетом объемов операций) значение, представляющее общерыночный обменный курс для конкретной пары валют (например, BTC/USD).

Существуют сотни приложений и веб-сайтов, на которых можно узнать текущий рыночный обменный курс. Ниже перечислены наиболее известные и посещаемые веб-сайты:

- Bitcoin Average (http://bitcoinaverage.com) – сайт предлагает легкочитаемый простой обзор средневзвешенных обменных курсов для каждой валюты;
- CoinCap (http://coincap.io) – сервис предоставляет списки рыночных оценок (капитализации) и обменные курсы для сотен криптовалют, включая и биткойн;
- Chicago Mercantile Exchange Bitcoin Reference Rate (http://www.cmegroup.com/trading/cf-bitcoin-reference-rate.html) – ставка-ориентир (reference rate), которая может использоваться для институциональной и договорной ставки, являющейся частью данных об инвестиционных выплатах CME.

Помимо разнообразных сайтов и приложений, большинство биткойн-кошельков автоматически вычисляет соотношение между биткойном и другими валютами. Джо воспользуется своим кошельком для автоматического преобразования цены перед отправкой биткойна Алисе.

Отправка и получение биткойна

Алиса решила поменять 10 долларов США на биткойн, чтобы не слишком рисковать своими деньгами при использовании этой новой для нее технологии. Она отдала Джо 10 долларов наличными, открыла свой кошелек (приложение Mycelium) и нажала кнопку **Receive** (Принять). После этого появился QR-код, представляющий первый биткойн-адрес Алисы.

Далее Джо нажал кнопку **Send** (Отправить) в своем кошельке на смартфоне, после чего появился экран с двумя полями ввода:

- биткойн-адрес получателя;
- отправляемая сумма в биткойнах (BTC) или в местной валюте (USD).

В поле ввода для биткойн-адреса есть маленькая пиктограмма, выглядящая как QR-код. Это позволяет Джо сканировать штриховой код с помощью видеокамеры смартфона, чтобы не вводить вручную достаточно длинный и слож-

ный биткойн-адрес Алисы. Касанием значка QR-кода Джо активизирует камеру смартфона и сканирует QR-код с экрана смартфона Алисы.

Теперь у Джо есть биткойн-адрес Алисы, определенный как получатель. Джо вводит сумму 10 долларов США, и кошелек выполняет автоматическое преобразование с учетом самого свежего обменного курса, полученного с онлайн-сервиса. На тот момент обменный курс составляет 100 долларов США за биткойн, так что 10 долларов равны 0.10 биткойна (BTC), или 100 миллибиткойнам (mBTC), как показано на снимке с экрана мобильного кошелька Джо (см. рис. 1.2).

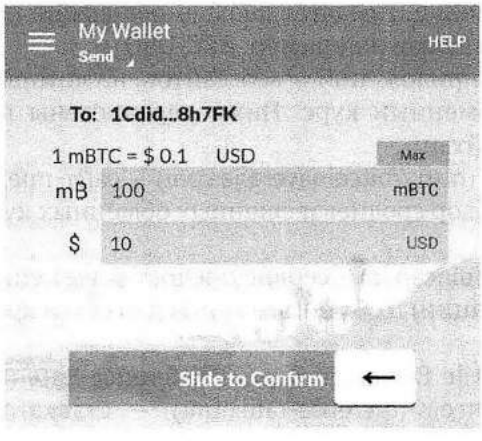


Рис. 1.2 ❖ Экран отправки денег мобильного биткойн-кошелька Airbitz

Затем Джо внимательно проверяет, правильно ли он ввел передаваемую сумму, поскольку после передачи денег ошибку исправить будет уже невозможно. После тщательнейшей проверки адреса и суммы Джо нажимает кнопку **Send** (Отправить), чтобы начать транзакцию. Мобильный биткойн-кошелек Джо создает транзакцию, которая передает 0.10 BTC на адрес, указанный Алисой, изымая денежные средства из кошелька Джо и подписывая текущую транзакцию с помощью секретных ключей Джо. Биткойн-сеть оповещается о том, что Джо выполнил авторизацию процедуры передачи определенной денежной суммы на новый адрес Алисы. Так как транзакция передается по пиринговому (peer-to-peer) протоколу, она быстро распространяется по биткойн-сети. Менее чем за секунду большинство надежно связанных между собой узлов этой сети принимает информацию о совершаемой транзакции и впервые видит адрес Алисы.

Тем временем кошелек Алисы постоянно «прослушивает» все транзакции, объявляемые в биткойн-сети, в поисках транзакции, целевой адрес которой совпадает с адресами кошельков Алисы. Через несколько секунд кошелек Джо завершает проведение транзакции, а в кошельке Алисы появляется оповещение о получении 0.10 биткойна (BTC).

Подтверждения

Сначала адрес Алисы обозначается в транзакции, выполняемой Джо, как «Unconfirmed» (Неподтвержденный). Это означает, что транзакция уже распространена по сети, но пока еще не записана в реестр транзакций биткойна, известного как блокчейн (blockchain). Для подтверждения транзакция обязательно должна быть включена в блок и добавлена в структуру данных блокчейна, а эта операция выполняется в среднем каждые 10 минут. В более привычных финансовых терминах такая операция называется клирингом (clearing). Более подробно операции распространения, проверки (валидации) и клиринга (подтверждения) будут рассматриваться в главе 10.

Теперь Алиса стала законной владелицей 0.10 биткойна, которые она может расходовать. В следующей главе мы рассмотрим ее первую покупку на биткойны и более подробно разберемся в технологиях, являющихся основой механизмов транзакций и распространения информации по сети.

Глава 2

Как работает биткойн

Транзакции, блоки, майнинг и блокчейн

Биткойн-система, в отличие от обычных банковских и платежных систем, основана на децентрализованных доверительных отношениях. Вместо центрального органа, облеченного соответствующими полномочиями и заслуживающего безусловного доверия, в биткойн-системе доверительные отношения формируются как эмерджентное (то есть приобретенное) свойство (emergent property) в результате многочисленных взаимодействий различных членов этой системы. В этой главе мы будем рассматривать биткойн-систему на верхнем уровне, прослеживая одну транзакцию, проходящую внутри системы, и наблюдая за тем, как она становится «заслуживающей доверия» и утверждается механизмом распределенного консенсуса биткойна, после чего записывается в структуру данных блокчейна, то есть в распределенный реестр всех транзакций. В последующих главах будут более подробно рассматриваться технологии, лежащие в основе механизма транзакций, сетевой среды и майнинга.

Общий обзор биткойн-системы

На обзорной схеме, показанной на рис. 2.1, мы видим, что биткойн-система состоит из пользователей с кошельками, содержащими ключи, из транзакций, распространяющихся по сети, и из майнеров, которые формируют (с помощью конкурентных вычислений) консенсус (согласование) структуры данных блокчейна, представляющей собой официальный и достоверный реестр всех транзакций.

Каждый пример в этой главе основан на транзакциях, действительно выполненных в биткойн-сети и имитирующих взаимодействие между пользователями (Джо, Алиса, Боб и Гопеш), заключающееся в передаче денежных средств из одного кошелька в другой. При прослеживании транзакции, проходящей через биткойн-сеть в структуру данных блокчейна, мы будем использовать сайт проводника блокчейна (blockchain explorer), чтобы наглядно представить каждый шаг. Проводник блокчейна – это веб-приложение, которое работает как поисковый механизм биткойн-системы, то есть позволяет вам искать адреса,

транзакции и блоки, а также наблюдать взаимоотношения узлов и потоки информации между ними.

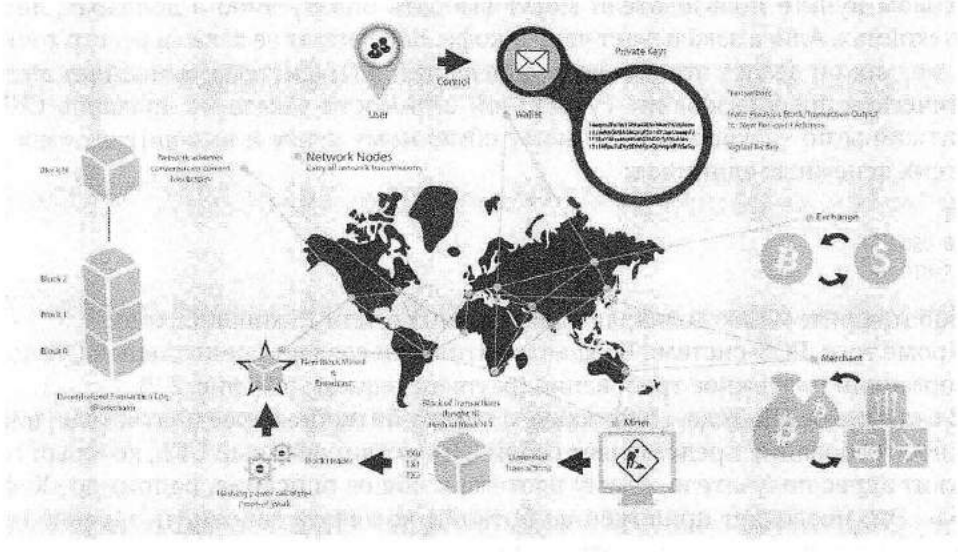


Рис. 2.1 ❖ Общая схема биткойн-системы

Самыми известными и широко используемыми проводниками блокчейна являются следующие:

- Bitcoin Block Explorer (https://blockexplorer.com);
- BlockCypher Explorer (https://live.blockcypher.com);
- blockchain.info (https:// blockchain.info);
- BitPay Insight (https://insight.bitpay.com).

В каждом из этих проводников есть функция поиска, с помощью которой можно узнать биткойн-адрес, хэш-значение транзакции, номер блока или хэш-значение блока и получить соответствующую информацию из биткойн-сети. Для каждого примера транзакции или использования блока мы будем получать URL, так что можно будет посмотреть, как выполняется операция, и изучить ее во всех подробностях.

Покупка чашки кофе

Алиса, с которой мы познакомились в предыдущей главе, – новый пользователь, который только что получил свои первые биткойны. В разделе «Получаем свой первый биткойн» главы 1 Алиса встретилась со своим другом Джо, чтобы обменять некоторую сумму наличными на биткойны. Транзакция, созданная Джо, пополнила кошелек Алисы на 0.10 BTC. Теперь Алиса выполнит свою первую розничную транзакцию, купив чашку кофе в кафетерии Боба в Пало Алто (Калифорния).

Кафе Боба совсем недавно начало принимать оплату биткойнами, добавив поддержку биткойна в свою систему POS-терминала (считывателя карт). Цены в кафе Боба объявлены в местной валюте (доллары США), но на контрольном кассовом пункте пользователи могут выбрать оплату либо в долларах, либо в биткойнах. Алиса заказывает чашку кофе, Боб вводит ее заказ в реестр точно так же, как он делает это для всех транзакций. POS-система выполняет автоматическое преобразование суммарной стоимости заказа из долларов США в биткойны по текущему рыночному обменному курсу и выводит стоимость в обеих денежных единицах:

```
Total:
$1.50 USD
0.015 BTC
```

Боб говорит: «С вас доллар пятьдесят центов, или 15 миллибитов».

Кроме того, POS-система Боба автоматически создает специальный QR-код, содержащий платежное требование (payment request) (см. рис. 2.2).

В отличие от QR-кода, содержащего только биткойн-адрес получателя, платежное требование представляет собой QR-закодированный URL, который содержит адрес получателя, сумму платежа и общее описание, например «Кафе Боба». Это позволяет приложению биткойн-кошелька заполнить заранее информационные поля, используемые при отправке платежа, с передачей пользователю описания, понятного человеку. Можно отсканировать этот QR-код с помощью приложения биткойн-кошелька, чтобы увидеть то, что должна видеть Алиса.

Рис. 2.2 ❖ QR-код платежного требования

Попробуйте отсканировать код на рис. 2.2 с помощью своего кошелька, чтобы увидеть адрес и сумму, но НЕ ПОСЫЛАЙТЕ ДЕНЬГИ ПО ЭТОМУ АДРЕСУ.

```
bitcoin:1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA?
amount=0.015&
label=Bob%27s%20Cafe&
message=Purchase%20at%20Bob%27s%20Cafe

Components of the URL
```

```
A bitcoin address: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
(Биткойн-адрес:)
The payment amount: "0.015"
(Сумма платежа:)
A label for the recipient address: "Bob's Cafe"
(Краткое сообщение по адресу получателя:)
A description for the payment: "Purchase at Bob's Cafe"
(Описание платежа:)
```

С помощью своего смартфона Алиса сканирует штриховой код и выводит его на дисплей. Смартфон показывает платеж в размере `0.0150` BTC в Кафе Боба (`Bob's Cafe`), потом Алиса нажимает кнопку **Send** (Отправить), чтобы подтвердить этот платеж. Через несколько секунд (это занимает приблизительно столько же времени, сколько авторизация кредитной карты) Боб видит в своем реестре полностью завершенную транзакцию.

В следующих разделах мы более подробно разберем эту транзакцию. Вы увидите, как кошелек Алисы создал ее, как созданная транзакция распространялась по сети, как ее проверили, наконец, как Боб может потратить эту сумму в последующих транзакциях.

Биткойн-сеть может проводить транзакции в дробных единицах стоимости, например от миллибиткойнов (1/1000 биткойна) до 1/100 000 000 (одной стомиллионной) биткойна, известной как «сатоши» (satoshi). В этой книге термин «биткойн» будет использоваться для обозначения количества биткойнов как денежных единиц: от минимальной – 1 сатоши – до общего максимального количества всех биткойнов (21 000 000), которые могут быть сгенерированы в результате майнинга.

Вы можете внимательно рассмотреть все подробности транзакции Алисы для кафе Боба в структуре данных блокчейна, используя один из сайтов проводника блокчейна (пример 2.1):

Пример 2.1 ❖ Просмотр транзакции Алисы с помощью сайта blockexplorer.com

```
https://blockexplorer.com/tx/
0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2
```

Транзакции биткойна

В упрощенном смысле транзакция сообщает сети о том, что владелец некоторого количества биткойнов подтвердил (с помощью процедуры авторизации) передачу этого количества биткойнов другому владельцу. После этого новый владелец также может расходовать биткойны, создавая другие транзакции, которые подтверждают (авторизуют) передачу биткойнов очередному владельцу и т. д. в цепочке прав владения.

Входные и выходные данные транзакции

Транзакции похожи на строки в книге бухгалтерского учета с двойной записью. Каждая транзакция содержит один или несколько элементов входных данных

(inputs), которые подобны дебетовым (расходным) записям на биткойновом счете. У транзакции также имеются некоторые выходные данные (outputs), подобные кредитовым (приходным) записям на биткойновом счете. Входные и выходные данные (дебет и кредит) не всегда одинаковы. Выходные данные могут добавлять несколько меньшую сумму, по сравнению с указанной во входных данных. Это небольшое различие объясняется подразумеваемым по умолчанию транзакционным сбором (transaction fee), то есть небольшой суммой, отчисляемой майнеру, который включает данную транзакцию в свой реестр. Транзакция биткойнов показана в виде записи в бухгалтерской книге на рис. 2.3.

Транзакция также содержит подтверждение (доказательство) права владения для каждой суммы биткойнов (входные данные), которая расходуется, в форме цифровой подписи владельца, подлинность которой может быть проверена любым членом сети независимо от других. В терминах биткойн-системы «расход» – это процедура подписи транзакции, которая передает некоторую стоимость из полностью завершенной предыдущей транзакции новому владельцу, идентифицируемому по биткойн-адресу.

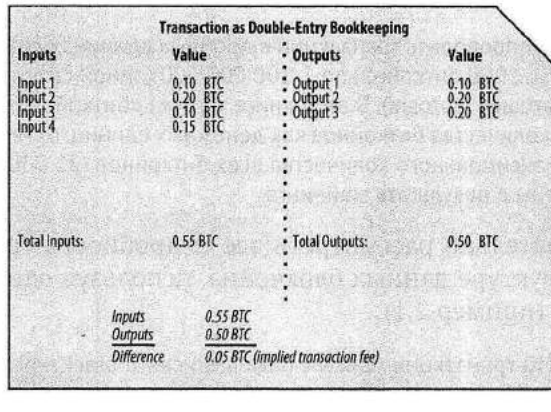


Рис. 2.3 ❖ Транзакция, представленная в форме книги бухгалтерского учета с двойной записью

Цепочки транзакций

Платеж Алисы в кафе Боба использует выходные данные предыдущей транзакции как свои входные данные. В предыдущей главе Алиса получила биткойны от своего друга Джо в обмен на наличные деньги. Та транзакция создала некоторую ценность, выраженную в биткойнах и защищенную («запертую») ключом Алисы. Ее новая транзакция для кафе Боба ссылается на предыдущую транзакцию как на входные данные и создает новые выходные данные для оплаты чашки кофе и получения сдачи. Эти транзакции образуют цепочку, в которой входные данные для самой последней транзакции соответствуют выходным данным, полученным из предшествующей транзакции. Ключ Алисы обеспечивает наличие подписи, которая разблокирует («отпирает») выход-

ные данные предыдущей транзакции, тем самым доказывая биткойн-сети, что именно Алиса владеет этими денежными средствами. Алиса связывает платеж за кофе с адресом Боба, создавая «платежное обязательство», выходные данные которого соответствуют платежному требованию, заверенному подписью Боба, чтобы потратить означенную сумму. Это действие представляет передачу стоимостной ценности между Алисой и Бобом. Цепочка транзакций от Джо к Алисе, затем к Бобу показана на рис. 2.4.

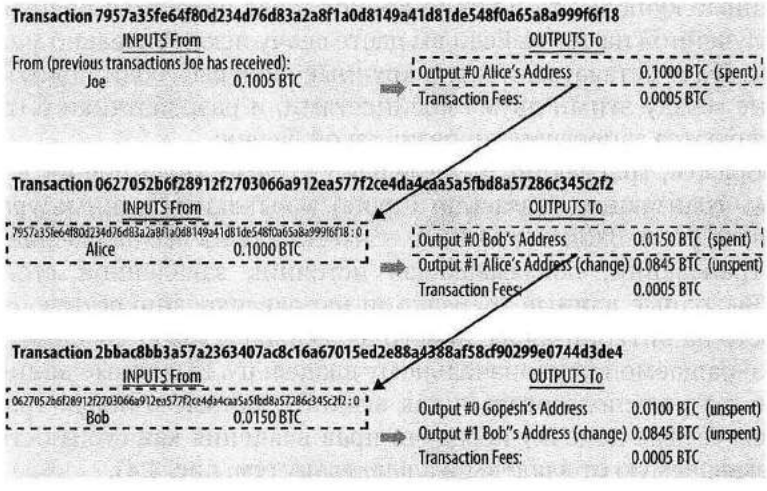


Рис. 2.4 ❖ Цепочка транзакций, в которой выходные данные одной транзакции становятся входными данными следующей транзакции

Получение сдачи

Многие транзакции биткойнов будут содержать выходные данные, ссылающиеся как на адрес нового владельца, так и на адрес текущего владельца, называемого адресом для получения сдачи (change address). Причина в том, что входные данные транзакции, подобно денежным банкнотам, не могут быть разделены. Если вы покупаете в магазине вещь стоимостью в 5 долларов, но при этом используете банкноту в 20 долларов, то вполне обоснованно рассчитываете получить 15 долларов сдачи. Точно такая же концепция применяется к входным данным транзакций биткойнов. Если вы купили вещь стоимостью в 5 биткойнов, но во входных данных использовалась сумма 20 биткойнов, то вам следовало отправить один фрагмент выходных данных с суммой 5 биткойнов владельцу магазина, а второй фрагмент данных с суммой 15 биткойнов вернуть себе как сдачу (чуть меньше из-за сбора за транзакцию). Важно отметить, что адрес для получения сдачи не обязательно должен совпадать с адресом, указанным во входных данных, и часто из соображений сохранения секретности в кошельке владельца создается новый адрес.

Кошельки могут использовать разнообразные стратегии объединения входных данных при формировании платежа, затребованного пользователем. Может объединяться множество небольших входных данных, или могут использоваться входные данные, равные или несколько бóльшие требуемого платежа. Если кошелек не может объединить входные данные таким образом, чтобы обеспечить точное соответствие сумме требуемого платежа плюс сбор за транзакцию, то кошелек должен сгенерировать соответствующую сдачу. Это очень похоже на обращение наличных денег. Если вы всегда носите с собой только крупные купюры, то в конце концов у вас наберется полный карман мелочи, полученной на сдачу. Если вы даете сдачу исключительно мелочью, то у вас всегда будут оставаться только крупные купюры. Люди подсознательно ищут баланс между этими двумя крайностями, и разработчики биткойн-кошельков стараются запрограммировать такой баланс.

Таким образом, транзакции перемещают стоимостную ценность из входных данных транзакции (trasaction inputs) в выходные данные транзакции (transaction outputs). Входные данные – это ссылка на выходные данные предыдущей транзакции, показывающая источник заявленной стоимостной ценности. Выходные данные транзакции направляют конкретную стоимостную ценность на биткойн-адрес нового владельца и могут содержать данные о сдаче, возвращаемой первоначальному владельцу. Выходные данные одной транзакции могут использоваться как входные данные в новой транзакции, создавая тем самым цепочку передачи прав владения как стоимостную ценность, перемещаемую от владельца к владельцу (см. рис. 2.4).

Общие формы транзакций

Наиболее общей формой транзакции является простой платеж с одного адреса на другой, в который часто включается некоторая сдача, возвращаемая первоначальному владельцу. При этом типе транзакции входные данные представлены одним фрагментом (один ввод), а выходные данные состоят из двух фрагментов (два вывода), как показано на рис. 2.5.

Рис. 2.5 ❖ Наиболее общая форма транзакции

Другой часто используемой формой является транзакция, объединяющая несколько фрагментов входных данных в единый фрагмент выходных данных (см. рис. 2.6). В реальной жизни этой операции соответствует обмен горсти монет и нескольких мелких купюр на одну более крупную банкноту. Подобные транзакции иногда генерируются приложениями кошельков, чтобы объединить множество мелких сумм, полученных в качестве сдачи при платежах.

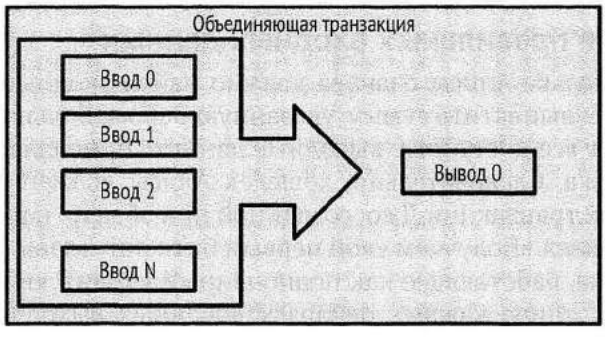


Рис. 2.6 ❖ Транзакция, объединяющая денежные средства

Еще одной формой, часто встречающейся в биткойн-реестре, является транзакция, распределяющая один фрагмент входных данных по нескольким фрагментам выходных данных, представляющих различных получателей (см. рис. 2.7). Этот тип транзакций иногда используется коммерческими организациями для распределения денежных средств, например при выдаче заработной платы нескольким сотрудникам.

Рис. 2.7 ❖ Транзакция, распределяющая денежные средства

Создание транзакции

Приложение кошелька Алисы содержит всю необходимую логику для выбора соответствующих входных и выходных данных при создании транзакции по

информации, предоставленной Алисой. Алиса должна указать только целевой адрес и сумму, а вся остальная работа выполняется приложением кошелька без участия пользователя. Важно, что приложение кошелька может создавать транзакции даже в режиме офлайн (то есть в режиме полного отключения от сети). Подобно тому, как мы выписываем чек дома и после этого отправляем его банку в конверте, транзакция не обязательно должна создаваться и подписываться при наличии активного соединения с биткойн-сетью.

Формирование правильных входных данных

Приложение кошелька Алисы сначала должно найти входные данные, которые позволили бы выплатить сумму, указанную Бобом. Большинство кошельков прослеживает все доступные выходные данные, принадлежащие адресам текущего кошелька. Следовательно, кошелек Алисы должен хранить копию выходных данных транзакции Джо, созданной при обмене наличных денег на биткойны (см. раздел «Получаем свой первый биткойн» главы 1). Приложение биткойн-кошелька, работающее как полноценный клиентский узел, действительно содержит копию каждых неизрасходованных выходных данных для каждой транзакции в структуре данных блокчейна. Это позволяет кошельку создавать входные данные транзакции, а также быстро проверять входящие транзакции как источник корректных входных данных. Но полноценный клиентский узел требует значительного дискового пространства, поэтому в большинстве пользовательских кошельков применяются упрощенные клиентские приложения, которые прослеживают только неизрасходованные выходные данные, принадлежащие текущему пользователю.

Если приложение кошелька не обеспечивает хранения копий неизрасходованных выходных данных транзакций, то оно может обратиться с запросом к биткойн-сети, чтобы получить эту информацию с помощью множества разнообразных прикладных программных интерфейсов (API), предоставляемых различными провайдерами, или обратившись к полноценному узлу сети с помощью вызова API-интерфейса. В примере 2.2 показан API-запрос, созданный в виде команды GET протокола HTTP, обращенной к заданному URL. Указанный URL возвращает все неизрасходованные выходные данные транзакции для некоторого адреса, предоставляя любому приложению информацию, необходимую для формирования входных данных транзакции, расходующей денежные средства. Здесь мы воспользуемся простой утилитой командной строки – HTTP-клиентом `curl` для получения ответа на наш запрос.

Пример 2.2 ❖ Поиск всех неизрасходованных выходных данных для биткойн-адреса Алисы

```
$ curl https://blockchain.info/unspent?active=1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK
{
  "unspent_outputs":[
    {
      "tx_hash":"186f9f998a5...2836dd734d2804fe65fa35779",
```

```
      "tx_index":104810202,
      "tx_output_n": 0,
      "script":"76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
      "value": 10000000,
      "value_hex": "00989680",
      "confirmations":0
    }
  ]
}
```

Ответ на запрос из примера 2.2 показывает один неизрасходованный фрагмент выходных данных (тот, который пока еще не реализован или «не погашен»), принадлежащий адресу Алисы `1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK`. Ответ включает ссылку на транзакцию, в которой содержится этот неизрасходованный фрагмент выходных данных (перевод от Джо), и указанную в нем сумму в сатоши (одна десятимиллионная биткойна), равную 0.10 биткойна. С помощью этой информации приложение кошелька Алисы может создать транзакцию для передачи этой суммы на адрес нового владельца.

Внимательно изучите транзакцию, передающую биткойны от Джо к Алисе (http://bit.ly/1tAeeGr)[1].

Как вы можете убедиться, кошелек Алисы содержит достаточное количество биткойнов, полученных как один неизрасходованный фрагмент выходных данных, для оплаты чашки кофе. В противном случае приложение кошелька Алисы, возможно, должно было бы начать поиски более мелких фрагментов неизрасходованных выходных данных, подобно тому, как роются в кошельке и в карманах в поисках мелочи, чтобы собрать сумму, необходимую для покупки чашки кофе. В обоих случаях, вероятнее всего, потребуется некоторая сумма сдачи, о которой мы более подробно поговорим в следующем разделе, где приложение кошелька будет создавать выходные данные транзакции (платежи).

Формирование выходных данных

Выходные данные транзакции формируются в виде скрипта, который создает закладное обязательство на передаваемую ценность, которое может быть погашено только представлением решения для этого скрипта. Проще говоря, выходные данные транзакции Алисы содержат скрипт, который говорит приблизительно следующее: «Эти выходные данные платежеспособны для того, кто может представить подпись с помощью ключа, соответствующего общедоступному адресу Боба». Поскольку лишь Боб обладает кошельком с ключами, соответствующими его адресу, только кошелек Боба сможет представить подпись для «погашения» заявленных выходных данных. Таким образом,

[1] Внимание: при попытке перейти по указанному адресу целевая страница не найдена, возможны перенаправления на ненадежные ресурсы. – *Прим. перев.*

Алиса «отдает в заклад» ценности, указанные в выходных данных, с запросом подписи Боба.

В эту транзакцию включен также еще один фрагмент выходных данных, так как денежная сумма в 0.10 BTC слишком велика для оплаты чашки кофе стоимостью 0.015 BTC. Алиса должна получить 0.085 BTC сдачи. Возврат сдачи Алисе создается ее кошельком в той же самой транзакции платежа Бобу. По сути, кошелек Алисы разделяет ее денежные средства на два платежа: один – Бобу, второй – возврат денег в кошелек Алисы. После этого она может использовать сдачу из выходных данных в следующей транзакции.

Наконец, за своевременную обработку транзакции в сети приложение кошелька Алисы отчисляет небольшую плату. Это неявная часть транзакции, она проявляется в виде разности между входными и выходными данными. Если вместо получения 0.085 сдачи Алиса создает только 0.0845 как второй фрагмент выходных данных, то у нее осталось бы 0.0005 BTC (половина милликойна). Сумма входных данных 0.10 BTC не полностью расходуется на два фрагмента выходных данных, потому что в сумме они составляют меньше 0.10. Итоговая разность представляет собой сбор за транзакцию (transaction fee), взимаемый майнером в качестве платы за проверку корректности и включение транзакции в блок, записываемый в структуру данных блокчейна.

Описанную выше транзакцию можно наблюдать во всех подробностях, используя веб-приложение проводника блокчейна, как показано на рис. 2.8.

Рис. 2.8 ❖ Транзакция Алисы, выполняющая оплату в кафе Боба

Внимательно изучите транзакцию, передающую биткойны от Алисы в кафе Боба (http://bit.ly/1u0FIGs)[1].

[1] Внимание: при попытке перейти по указанному адресу целевая страница не найдена, возможны перенаправления на ненадежные ресурсы. – *Прим. перев.*

Добавление транзакции в реестр

Транзакция, созданная приложением кошелька Алисы, имеет длину 258 байтов и содержит все необходимое для подтверждения права владения денежными средствами и передачи их новым владельцам. Теперь эта транзакция непременно должна быть передана в биткойн-сеть, где она станет частью структуры данных блокчейна. В следующем разделе мы увидим, как транзакция становится частью нового блока и как этот блок «майнится». В конце главы мы рассмотрим, как новый блок после добавления в структуру данных блокчейна постепенно зарабатывает доверие в сети по мере добавления других блоков.

Передача транзакции

Поскольку транзакция содержит всю информацию, необходимую для ее обработки, не имеет значения, как и где эта транзакция передается в биткойн-сеть. Биткойн-сеть – это пиринговая сеть, в которой каждый биткойн-клиент соединен с несколькими другими биткойн-клиентами. Основная задача биткойн-сети – распространение транзакций и блоков между всеми членами сети.

Как происходит распространение

Любая система, такая как сервер, десктоп-приложение или кошелек, участвующая в биткойн-сети, то есть «говорящая на языке» протокола биткойна, называется биткойн-узлом (bitcoin node). Приложение кошелька Алисы может отправлять новую транзакцию на любой биткойн-узел через установленное соединение практически любого типа: кабельное, Wi-Fi, мобильное и т. д. Ее биткойн-кошелек не обязан устанавливать прямое соединение с биткойн-кошельком Боба, и нет никакой необходимости обязательно использовать подключение к Интернету, предлагаемое кафе Боба, хотя оба этих способа также возможны. Любой биткойн-узел, принимающий допустимую транзакцию, которая до сего момента была ему неизвестна, немедленно перенаправляет ее всем узлам, с которыми он соединен, такая методика распространения известна под названием лавинная адресация (flooding). Таким образом, транзакция быстро распространяется по пиринговой сети и приходит на большинство узлов за несколько секунд.

Как это видит Боб

Если приложение биткойн-кошелька Боба напрямую соединено с приложением кошелька Алисы, то приложение кошелька Боба может стать первым узлом, принявшим рассматриваемую транзакцию. Но даже если кошелек Алисы отправляет транзакцию через другие узлы, через несколько секунд эта транзакция дойдет и до кошелька Боба. Кошелек Боба сразу же определит транзакцию Алисы как поступивший платеж, так как транзакция содержит выходные данные, подтверждаемые ключами Боба. Кроме того, приложение кошелька Боба может независимо от прочих узлов проверить, правильно ли сформирована эта транзакция, использует ли она ранее неизрасходованные входные данные и содержит ли достаточную сумму для оплаты сбора за транзакцию, чтобы по-

явилась возможность включения ее в следующий блок. В этот момент Боб может решить с небольшой долей риска, что транзакция в ближайшем будущем будет включена в блок и подтверждена.

Самая распространенная ошибка в понимании транзакций биткойнов – неправильное представление о том, что транзакции обязательно должны быть «подтверждены» после 10 минут ожидания появления нового блока или через 60 минут для шести полноценных подтверждений. Несмотря на то что подтверждения гарантируют принятие конкретной транзакции всей сетью в целом, указанные выше задержки вовсе не обязательны для самых дешевых товаров, таких как чашка кофе. Продавец может принять допустимую транзакцию с небольшой суммой без подтверждений, причем риск здесь не больше, чем оплата кредитной картой без идентификатора или подписи, принимаемая в наше время розничными продавцами десятки, а то и сотни раз в день.

Майнинг биткойнов

Итак, транзакция Алисы теперь распространяется по биткойн-сети. Она не станет частью структуры данных блокчейна до тех пор, пока не будет проверена и включена в блок в результате выполнения процесса, называемого майнингом (mining). Более подробно этот процесс будет рассматриваться в главе 10. Доверительные отношения в биткойн-системе основаны на математических вычислениях. Транзакции объединяются в блоки (blocks), доказательство корректности которых требует огромного объема вычислений, но для проверки принятого блока необходим относительно небольшой объем вычислений. Процесс майнинга служит достижению двух целей в биткойн-системе:

- узлы майнинга проверяют правильность (validate) всех транзакций на соответствие правилам консенсуса (consensus rules) биткойн-системы. Следовательно, майнинг обеспечивает защиту транзакций биткойнов, отвергая недопустимые или неправильно сформированные транзакции;
- майнинг создает новые биткойны в каждом блоке почти так же, как центральный банк печатает новые денежные купюры. Количество биткойнов, создаваемых в одном блоке, ограничено и уменьшается со временем в соответствии с общим планом ограниченной эмиссии.

Майнинг позволяет достичь почти идеального баланса между стоимостью и вознаграждением (компенсацией). Для решения математических задач в процессе майнинга расходуется электроэнергия. Успешный майнер будет получать вознаграждение или компенсацию (reward) в форме новых биткойнов и сборов за выполнение транзакций. Но вознаграждение можно получать только в том случае, если майнер действительно правильно проверил все транзакции на соответствие правилам консенсуса (consensus). Этот динамический баланс обеспечивает защиту биткойнов без какого-либо центрального органа управления.

Неплохой аналогией для описания процесса майнинга является крупномасштабное соревнование по решению головоломок судоку (sudoku), которое возобновляется всякий раз, когда кто-нибудь находит решение, при этом сложность головоломок автоматически регулируется таким образом, чтобы на поиск решения требовалось около 10 минут. Представьте себе гигантскую головоломку судоку размером в несколько тысяч строк и столбцов. Если я покажу вам решенную головоломку, то вы сможете проверить правильность моего решения достаточно быстро. Но если головоломка содержит несколько заполненных квадратов, а остальные пусты, то решение потребует огромного объема работы. Сложность судоку можно регулировать изменением ее размера (увеличивая или уменьшая количество строк и столбцов), но проверка решения может оставаться относительно простой задачей даже при больших размерах. Головоломка, используемая в биткойн-системе, основана на криптографических хэш-значениях и обладает похожими характеристиками: решение ее несравнимо труднее, чем проверка (которая значительно проще), а сложность можно регулировать.

В разделе «Варианты использования биткойнов, пользователи и их истории» главы 1 мы познакомились с Цзином, предпринимателем из Шанхая. Цзин создал ферму майнинга (mining farm), то есть предприятие, использующее тысячи компьютеров, предназначенных специально для майнинга и для конкуренции за вознаграждение. С интервалами, равными приблизительно 10 минутам, майнинговые компьютеры Цзина вступают в состязание с тысячами подобных систем в глобальном процессе поиска решения для блока транзакций. Поиск такого решения, или так называемое доказательство выполнения работы (Proof-of-Work, PoW), требует квадрильонов операций хэширования в секунду по всей биткойн-сети в целом. Алгоритм доказательства выполнения работы (PoW-алгоритм) подразумевает повторяющееся выполнение операции хэширования заголовка блока и случайного числа с помощью криптографического алгоритма SHA256 до тех пор, пока вычисленное значение не совпадет полностью с предварительно определенным образцом (шаблоном). Первый майнер, нашедший правильное решение, побеждает в текущем раунде состязания и оповещает всех членов сети о включении соответствующего блока в структуру данных блокчейна.

Цзин начал заниматься майнингом в 2010 году, используя высокопроизводительный настольный компьютер, чтобы вычислять соответствующее доказательство выполнения работы для новых блоков. Постепенно к биткойн-сети начали присоединяться новые майнеры, и сложность задачи быстро увеличивалась. Вскоре Цзин и другие майнеры перешли на более производительное оборудование, например на дорогостоящие специализированные устройства обработки графики (GPU), то есть видеокарты, используемые в игровых компьютерах или игровых консолях. Во время написания этой книги сложность задач возросла настолько, что процесс майнинга стал окупаться только при применении интегральных схем специального назначения (application-

specific integrated circuits, ASIC), в которых сотни алгоритмов майнинга зашиты непосредственно в аппаратуру и работают в параллельном режиме на одном чипе. Компания Цзина также участвует в пуле майнинга (mining pool), который во многом напоминает лотерейный пул, позволяющий нескольким участникам объединять свои усилия и затраты и совместно пользоваться выигрышами. Сейчас компания Цзина работает как крупное предприятие, содержащее тысячи ASIC-майнеров для круглосуточного майнинга биткойнов. Компания оплачивает расход электроэнергии, продавая биткойны, сгенерированные в процессе майнинга, при этом извлекая некоторую прибыль из общего дохода.

Майнинг транзакций в блоках

Новые транзакции постоянно приходят в сеть из кошельков пользователей и от других приложений. Как только эти транзакции попадают в поле зрения узлов биткойн-сети, они добавляются во временный пул непроверенных транзакций, поддерживаемый каждым узлом. Когда майнеры создают новые блоки, они добавляют непроверенные транзакции из пула в новый блок, затем пытаются доказать корректность этого нового блока с помощью алгоритма майнинга (PoW). Более подробно процесс майнинга описан в главе 10.

Транзакции добавляются в новый блок с назначением приоритета в первую очередь по наибольшей сумме сбора за транзакцию и по нескольким другим критериям. Каждый майнер начинает процесс майнинга нового блока транзакций сразу же после получения предыдущего блока из сети, понимая, что он проиграл только что завершившийся раунд состязания. Майнер немедленно создает новый блок, заполняет его транзакциями, включает в блок отпечаток предыдущего блока и начинает вычисление по алгоритму доказательства выполнения работы для нового блока. Каждый майнер включает в свой блок особую транзакцию, в которой назначает оплату своему биткойн-адресу как вознаграждение за этот блок (в настоящее время 12.5 вновь созданного биткойна) плюс сумма сборов за все транзакции, включенные в этот блок. Если майнер находит решение, подтверждающее корректность блока, то он «выигрывает» назначенное вознаграждение, потому что его успешно созданный блок добавляется в общую структуру данных блокчейна, а оплаченные транзакции, включенные в этот блок, становятся утвержденными (проведенными). Цзин, участвующий в пуле майнинга, настроил свое программное обеспечение на создание новых блоков, которые передают вознаграждение на адрес пула. Поэтому полученное совместными усилиями вознаграждение распределяется между Цзином и другими майнерами пропорционально объему работы, который они фактически выполнили в прошедшем раунде.

Транзакция Алисы была принята сетью и помещена в пул непроверенных транзакций. После проверки программным обеспечением майнинга

транзакция была включена в новый блок, называемый блоком-кандидатом (candidate block), сгенерированный пулом майнинга Цзина. Все майнеры, участвующие в этом пуле, немедленно начали вычисления по алгоритму доказательства выполнения работы для блока-кандидата. Приблизительно через пять минут после того, как транзакция была передана из кошелька Алисы, один из ASIC-майнеров Цзина нашел решение для блока-кандидата и объявил об этом в сети. После того как прочие майнеры проверили корректность победившего блока, они сразу включились в состязание по генерации следующего блока.

Победивший блок Цзина стал частью структуры данных блокчейна как блок под номером #277316, содержащий 420 транзакций, в том числе и транзакцию Алисы. Блок, в котором находится транзакция Алисы, теперь считается «подтверждением» правильности этой транзакции.

Вы можете увидеть блок, который содержит транзакцию Алисы (https://blockchain.info/block-height/277316).

Приблизительно через 19 минут другим майнером был добавлен новый блок под номером #277317. Поскольку этот новый блок создан поверх блока #277316, содержащего транзакцию Алисы, он увеличил объем вычислений в структуре блокчейна, следовательно, укрепил доверие к этой и другим транзакциям в блоке. Каждый блок, помещенный в результате майнинга поверх предыдущего, содержит счетчики транзакций как дополнительное подтверждение легальности транзакции Алисы. Так как блоки размещаются один поверх другого своеобразным «столбиком», сложность отмены этой транзакции постоянно возрастает по экспоненте, следовательно, транзакция получает все больше и больше доверия в сети.

На схеме рис. 2.9 можно видеть блок #277316, содержащий транзакцию Алисы. Ниже него находятся 277 316 блоков (начиная с блока #0), связанных друг с другом в цепочку блоков (blockchain – дословно «цепочка блоков») до самого первого блока #0, называемого первичным блоком (genesis block). Со временем «высота столбика» блоков увеличивается, соответственно, возрастает и сложность вычисления для каждого блока и для цепочки в целом. Блоки, созданные в результате майнинга после блока, содержащего транзакцию Алисы, становятся дополнительной гарантией, поскольку наращивают объем вычислений в постоянно увеличивающей свою длину цепочке. По соглашению любой блок с количеством подтверждений, большим шести, считается не подлежащим отмене, так как при отмене потребуется огромный объем вычислений для аннулирования и пересчета по шести блокам. Более подробно процесс майнинга и формирования доверительного отношения в ходе этого процесса мы рассмотрим в главе 10.

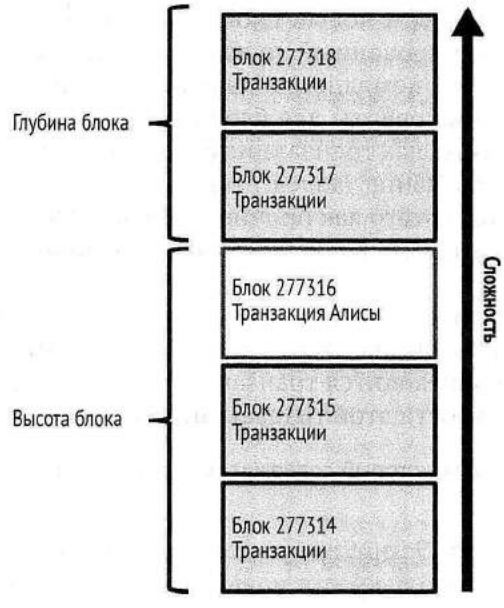


Рис. 2.9 ❖ Транзакция Алисы, включенная в блок #277316

Расходование транзакции

Теперь, когда транзакция Алисы включена в структуру данных блокчейна как часть одного из блоков, она становится частью распределенного реестра биткойнов и ее видят все биткойн-приложения. Каждый биткойн-клиент может независимо от других проверить эту транзакцию на корректность и платежеспособность. Полноценные узлы-клиенты могут проследить источник поступления денежных средств от момента генерации соответствующего биткойна в блоке, постепенное продвижение его из транзакции в транзакцию до тех пор, пока указанная сумма не дойдет до адреса Боба. Упрощенные клиенты могут выполнить операцию, называемую упрощенной проверкой (верификацией) платежа (см. раздел «Узлы упрощенной верификации платежей (SPV)» в главе 8), подтверждающую, что транзакция находится в структуре данных блокчейна и после нее добавлено несколько блоков в результате майнинга, тем самым давая гарантию того, что майнеры приняли эту транзакцию как легальную.

Теперь Боб может расходовать выходные данные (денежные средства) этой и других транзакций. Например, Боб может заплатить подрядчику или поставщику, передавая стоимость чашки кофе, купленной Алисой, в качестве платежа новым владельцам. Но более вероятно, что биткойн-приложение Боба объединит множество мелких платежей в более крупный платеж, возможно, собрав все поступления биткойнов за день в единую транзакцию. При этом различные

платежи объединяются в один фрагмент выходных данных (на одном адресе). Схему объединяющей транзакции см. на рис. 2.6.

Когда Боб расходует платежи, полученные от Алисы и прочих покупателей, он увеличивает длину цепочки транзакций. Предположим, что Боб платит своему веб-дизайнеру Гопешу из Бангалора за создание новой страницы веб-сайта. После этого цепочка транзакций будет выглядеть так, как показано на рис. 2.10.

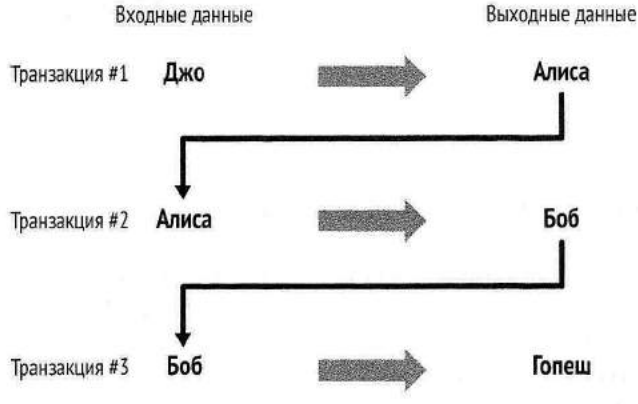


Рис. 2.10 ❖ Транзакция Алисы
как часть цепочки транзакций от Джо до Гопеша

В этой главе мы рассмотрели, как транзакции формируют цепочку, которая перемещает стоимостные ценности от владельца к владельцу. Мы также подробно проследили транзакцию Алисы от момента создания в ее кошельке, дальнейшее продвижение по биткойн-сети к майнерам, которые записали эту транзакцию в структуру данных блокчейна. Далее мы будем изучать конкретные технологии, лежащие в основе программного обеспечения кошельков, адресов, подписей, транзакций, сетевой среды и, конечно, майнинга.

Глава 3

Bicoin Core: эталонная реализация

Биткойн – это проект с открытым исходным кодом (open source), и все исходные коды доступны под открытой лицензией MIT, то есть их можно скачивать бесплатно и использовать для любых целей. Открытый исходный код означает нечто большее, чем просто бесплатное использование. Это означает, что программное обеспечение биткойна разрабатывается открытым для всех сообществом добровольцев. Сначала это сообщество состояло из одного Сатоши Накамото. Но к 2016 году в исходные коды биткойна внесли свой вклад (в той или иной форме) более 400 человек, а около десятка разработчиков отдавали биткойну все свое рабочее время, кроме того, несколько десятков занимались разработкой ПО биткойна как частично занятые сотрудники. Каждый может внести свой вклад в разработку кода, в том числе и вы.

При создании биткойна Сатоши Накамото полностью завершил разработку программного обеспечения до написания своей статьи, представленной в приложении А. Сатоши хотел убедиться в том, что все работает надлежащим образом, прежде чем писать об этом. Первая реализация, в дальнейшем известная под названием Bitcoin, или Satoshi client, была существенно изменена и улучшена. Реализация Сатоши постепенно развилась в то, что сейчас называют Bitcoin Core, чтобы отличать эту версию от других совместимых реализаций. Bitcoin Core – это эталонная реализация (reference implementation) биткойн-системы, то есть авторитетный источник, в котором каждый компонент технологии должен быть полноценно реализован. В Bitcoin Core реализованы все аспекты биткойна, включая кошельки, механизм транзакций и механизм проверки (валидации) блока, а также полноценный сетевой узел в пиринговой биткойн-сети.

Несмотря на то что в Bitcoin Core включена эталонная реализация кошелька, она не предназначена для реально используемых кошельков пользователей или приложений. Прикладным разработчикам рекомендуется создавать кошельки на основе современных стандартов, таких как BIP-39 и BIP-32 (см. разделы «Мнемонические кодовые слова (BIP-39)» и «HD-кошельки (BIP-32/BIP-44)» в главе 5). BIP означает Bitcoin Improvement Proposal (Предложение по улучшению биткойна).

На рис. 3.1 показана схема архитектуры Bitcoin Core.

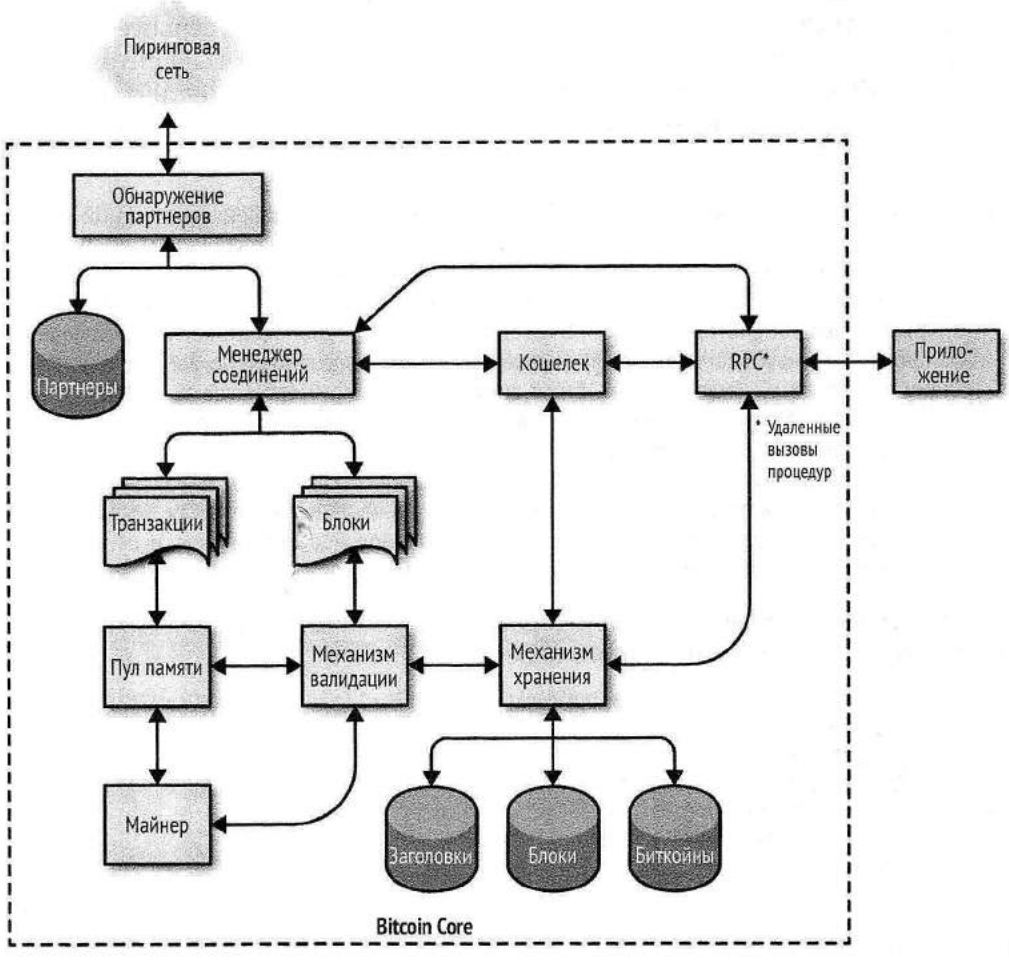


Рис. 3.1 ❖ Архитектура Bitcoin Core (источник: Эрик Ломброзо (Eric Lombrozo))

Среда разработки биткойна

Если вы разработчик ПО, то, вероятно, захотите установить среду разработки со всеми инструментальными средствами, библиотеками и программами поддержки и сопровождения для написания биткойн-приложений. В этой главе, насыщенной техническими подробностями, мы рассмотрим процесс разработки постепенно, шаг за шагом. Если излагаемый материал покажется слишком трудным для понимания (и если вы не установили среду разработки), то вы спокойно можете перейти к чтению следующей главы, в которой технической информации не так много.

Компиляция Bitcoin Core из исходных кодов

Исходные коды Bitcoin Core можно загрузить в виде ZIP-архива или клонировать соответствующую ветвь авторизованного репозитория на сервисе GitHub. На странице GitHub bitcoin (https://github.com/bitcoin/bitcoin) выберите пункт Download ZIP в боковой панели. Другой способ – использование интерфейса командной строки git для создания локальной копии исходного кода биткойна в вашей системе.

Во многих примерах этой главы мы будем использовать интерфейс командной строки операционной системы, известный также как командная оболочка (shell), доступный с помощью приложения типа терминал (terminal). Командная оболочка выводит подсказку (промпт; prompt), вы вводите команду с клавиатуры, затем командная оболочка отвечает, выводя некоторый текст и очередную подсказку для ввода вашей следующей команды. В разных системах промпт может выглядеть по-разному, но во всех последующих примерах он будет обозначаться символом $. В примерах, где вы видите текст после символа $, сам этот символ вводить не надо, а необходимо ввести следующую непосредственно за ним команду, затем нажать клавишу **Enter**, чтобы выполнить команду. В большинстве примеров строка (или несколько строк) под каждой командой представляет собой ответ операционной системы на введенную команду. Когда строка начинается с очередного промпта $, это означает ввод новой команды, и вы должны будете воспроизвести данный процесс.

В приведенном выше примере используется команда `git` для создания локальной копии («клона») исходного кода:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 66193, done.
remote: Total 66193 (delta 0), reused 0 (delta 0), pack-reused 66193
Receiving objects: 100% (66193/66193), 63.39 MiB | 574.00 KiB/s, done.
Resolving deltas: 100% (48395/48395), done.
Checking connectivity... done.
$
```

Git – это самая широко используемая распределенная система управления версиями, весьма важная часть инструментария каждого разработчика ПО. Возможно, вам придется установить программный пакет git или графический пользовательский интерфейс для git в своей операционной системе, если он не был установлен ранее.

После завершения операции клонирования командой `git` вы получаете полную локальную копию репозитория исходного кода в каталоге *bitcoin*. Перейдите в этот каталог, введя следующую команду после подсказки-промпта:

```
$ cd bitcoin
```

Выбор версии Bitcoin Core

По умолчанию локальная копия синхронизируется с самым свежим исходным кодом, который может быть нестабильным или представлять бета-версию

биткойна. Перед компиляцией кода выберите конкретную версию, явно указав номер релиза *tag*. Это позволит синхронизировать локальную копию с указанным снимком репозитория исходного кода, определяемого по ключевому слову *tag*. Теги используются разработчиками для обозначения каждого релиза кода номером версии. Сначала, чтобы увидеть все доступные теги, воспользуемся командой `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 показаны все доступные версии биткойна. По соглашению предварительные версии или релиз-кандидаты (release candidates), предназначенные для тестирования, помечаются суффиксом rc. Стабильные версии для реальной эксплуатации не имеют суффикса. Из приведенного выше списка выберите самую свежую стабильную версию, на время написания книги это была версия v0.11.2. Для синхронизации локальной копии кода с этой версией воспользуемся следующей командой:

```
$ git checkout v0.11.2
HEAD is now at 7e27892... Merge pull request #6975
```

Можно убедиться в том, что получена требуемая версия, проверив ее с помощью команды, показанной ниже:

```
$ git status
HEAD detached at v0.11.2
nothing to commit, working directory clean
```

Конфигурирование компилируемой версии Bitcoin Core

В комплект исходных кодов включена документация, которую можно найти в нескольких файлах. Основную документацию в файле *README.md*, расположенном в каталоге *bitcoin*, можно просмотреть командой `more README.md`, если ввести ее после промпта и использовать клавишу пробела для перехода к следующей странице. В этой главе мы скомпилируем биткойн-клиент для работы в командной строке, также известный под названием `bitcoind` в ОС Linux. Изучите инструкции по компиляции клиента для командной строки `bitcoind` на своей платформе с помощью команды `more doc/build-unix.md`. Инструкции для операционных систем macOS и Windows можно найти в каталоге doc, в файлах *build-osx.md* и *build-windows.md* соответственно.

Внимательно прочитайте предварительные требования к процедуре компиляции и сборки, приведенные в первой части документации. Здесь указаны библиотеки, которые должны быть установлены в системе до начала компиляции биткойна. Если какое-либо из предварительных требований не будет выполнено, то компиляция аварийно завершится с сообщением об ошибке. Если такое произошло, то вы можете установить недостающую библиотеку и еще раз запустить процедуру компиляции и сборки. Предположим, что все необходимые компоненты установлены, и процесс начинается с генерации набора скриптов сборки с помощью специального скрипта *autogen.sh*.

В процессе сборки Bitcoin Core система autogen/configure/make применяется с версии 0.9. Более старые версии использовали простой Makefile, и процедура сборки немного отличалась от показанной в следующем примере. Выполняйте инструкции для той версии, которую намерены скомпилировать. Вероятнее всего, введенная в версии 0.9 система autogen/configure/make станет системой сборки для всех будущих версий кода, поэтому именно ее работа показана в следующих примерах.

```
$ ./autogen.sh
...
glibtoolize: copying file 'build-aux/m4/libtool.m4'
glibtoolize: copying file 'build-aux/m4/ltoptions.m4'
glibtoolize: copying file 'build-aux/m4/ltsugar.m4'
glibtoolize: copying file 'build-aux/m4/ltversion.m4'
...
configure.ac:10: installing 'build-aux/compile'
configure.ac:5: installing 'build-aux/config.guess'
configure.ac:5: installing 'build-aux/config.sub'
configure.ac:9: installing 'build-aux/install-sh'
configure.ac:9: installing 'build-aux/missing'
Makefile.am: installing 'build-aux/depcomp'
...
```

Скрипт *autogen.sh* создает набор скриптов автоматического конфигурирования, которые опрашивают целевую систему, чтобы получить правильные параметры конфигурации и убедиться в том, что все необходимые для компиляции кода библиотеки установлены. Самым важным в этом наборе является скрипт `configure`, предлагающий разнообразные варианты для настройки процесса компиляции и сборки. Чтобы увидеть все возможные ключи и опции настройки, выполните следующую команду:

```
$ ./configure --help
`configure' configures Bitcoin Core 0.11.2 to adapt to many kinds of systems.

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

...
Optional Features:
(Дополнительные возможности:)
  --disable-option-checking  ignore unrecognized --enable/--with options
                             (игнорировать нераспознанные ключи --enable/--with)
```

```
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
                          (не включать функцию FEATURE (аналог --enable-FEATURE=no))
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
                          (включить функцию FEATURE [ARG=yes])

  --enable-wallet         enable wallet (default is yes)
                          (разрешить использование кошелька (по умолчанию разрешен))

  --with-gui[=no|qt4|qt5|auto]
...
```

Скрипт `configure` позволяет разрешить или запретить некоторые функциональные возможности программы `bitcoind` с помощью флагов `--enable-FEATURE` и `--disable-FEATURE`, где `FEATURE` заменяется именем конкретной функции, указанной в выводе вспомогательной информации для скрипта. В этой главе будут выполнены компиляция и сборка клиента `bitcoind` со всеми функциональными возможностями, принятыми по умолчанию. Мы не будем использовать флаги конфигурации, но вам следует внимательно изучить их описание, чтобы понять, какие дополнительные функции входят в состав этого клиента. Если конфигурирование выполняется в учебной среде, то по условиям учебной компьютерной лаборатории может потребоваться установка приложений в личном домашнем каталоге (то есть с использованием флага `--prefix=$HOME`).

Ниже приведено описание нескольких полезных флагов и опций, которые изменяют поведение конфигурационного скрипта, определенное по умолчанию.

`--prefix=$HOME`
: Позволяет изменить определенное по умолчанию местоположение устанавливаемой программы (обычно это каталог */usr/local/*). Значение `$HOME` дает возможность устанавливать все компоненты и программы в свой домашний каталог, но можно указать любой другой существующий путь.

`--disable-wallet`
: Используется для запрещения сборки эталонной реализации кошелька.

`--with-incompatible-bdb`
: Если выполняются компиляция и сборка кошелька, то этот флаг разрешает использование несовместимой версии библиотеки Berkeley DB.

`--with-gui=no`
: Отключает сборку графического пользовательского интерфейса, при которой требуется библиотека Qt. В результате выполняется сборка только сервера и утилиты командной строки биткойна.

После этого запускается скрипт `configure` для автоматического определения наличия всех требуемых библиотек и создания скрипта сборки, специализированного для конкретной системы:

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
...
[далее следует несколько страниц с результатами тестов конфигурации]
...
$
```

Если процесс завершился успешно, то команда `configure` создает специализированные скрипты сборки, позволяющие скомпилировать и собрать `bitcoind`. Если какие-либо библиотеки не найдены или обнаружены какие-либо проблемы, то команда `configure` завершится с ошибкой и не создаст скриптов сборки. Наиболее вероятные проблемы – отсутствие необходимой библиотеки или несовместимая версия такой библиотеки. В этом случае еще раз внимательно прочитайте документацию по сборке, проверьте и установите все требуемые библиотеки. Затем запустите скрипт `configure` повторно и посмотрите, помогло ли это устранить проблему.

Сборка выполняемых файлов Bitcoin Core

После процедуры конфигурации будет выполнена компиляция исходных кодов, причем для завершения этого процесса может потребоваться около часа, в зависимости от скорости центрального процессора и объема доступной оперативной памяти. Во время компиляции сообщения о ее ходе будут выводиться через каждые несколько секунд (или через каждые несколько минут), или вы получите сообщение об ошибке, если что-то пошло не так. При ошибке процесс компиляции прерывается, но его можно возобновить в любое время, введя команду `make`. Для начала компиляции выполняемых файлов приложения выполните следующую команду:

```
$ make
Making all in src
  CXX      crypto/libbitcoinconsensus_la-hmac_sha512.lo
  CXX      crypto/libbitcoinconsensus_la-ripemd160.lo
  CXX      crypto/libbitcoinconsensus_la-sha1.lo
  CXX      crypto/libbitcoinconsensus_la-sha256.lo
  CXX      crypto/libbitcoinconsensus_la-sha512.lo
  CXX      libbitcoinconsensus_la-hash.lo
  CXX      primitives/libbitcoinconsensus_la-transaction.lo
  CXX      libbitcoinconsensus_la-pubkey.lo
  CXX      script/libbitcoinconsensus_la-bitcoinconsensus.lo
  CXX      script/libbitcoinconsensus_la-interpreter.lo

[... множество сообщений о ходе компиляции ...]

$
```

Если процесс завершился успешно, то Bitcoin Core скомпилирован полностью. Заключительный этап – установка всех выполняемых файлов в системе с помощью команды `sudo make install`. Возможно, потребуется ввести пароль пользователя, так как для выполнения этой команды необходимы привилегии администратора (суперпользователя):

```
$ 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
...
$
```

По умолчанию `bitcoind` устанавливается в каталог */usr/local/bin*. Проверить правильность установки можно с помощью запроса системного пути к выполняемым файлам, как показано ниже:

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

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

Запуск узла Bitcoin Core

Пиринговая сеть биткойна состоит из сетевых узлов (nodes), работа которых поддерживается в основном добровольными участниками и некоторыми бизнес-организациями, которые создают биткойн-приложения. Этим активным биткойн-узлам разрешен прямой и полномочный доступ к структуре данных блокчейна в биткойн-системе с возможностью создания локальной копии всех транзакций, независимо от других проверяемой в собственной системе каждого узла. После ввода узла в эксплуатацию вы не должны полагаться на какую-либо третью сторону для проверки транзакций. Более того, после запуска активного биткойн-узла вы вносите определенный вклад в биткойн-сеть, делая ее более надежной и устойчивой.

Но ввод узла в реальную эксплуатацию требует постоянного подключения к сети системы с вычислительными ресурсами, достаточными для обработки всех биткойн-транзакций. Если вы намерены выполнять индексирование всех транзакций и хранить полную копию структуры данных блокчейна, то для этого может потребоваться огромный объем дискового пространства и оперативной памяти. В конце 2016 года для узла с поддержкой полной индексации необходимо было иметь 2 Гб оперативной памяти и 125 Гб дискового пространства с возможностью наращивания этих ресурсов. Биткойн-узлы также передают и принимают блоки и транзакции биткойнов, используя для этого пропускную

способность Интернета. Если ваше соединение с Интернетом ограничено по определенному объему трафика или с увеличением скорости тариф повышается, то, возможно, не следует запускать биткойн-узел на таком соединении или необходимо организовать его работу так, чтобы ограничить его производительность (пропускную способность) (см. пример 3.2).

Bitcoin Core сохраняет полную копию данных блокчейна по умолчанию с записями о каждой транзакции, выполненной в биткойн-сети с момента ее появления в 2009 году. Этот набор данных имеет размер в десятки гигабайтов, поэтому загружается постепенно в течение нескольких дней или даже недель, в зависимости от ресурсов вашего компьютера и пропускной способности интернет-соединения. Bitcoin Core не сможет обрабатывать транзакции и обновлять балансы учетных записей до тех пор, пока не будет полностью загружена вся структура данных блокчейна. Вы должны быть абсолютно уверены в том, что действительно имеете достаточное дисковое пространство, соединение с высокой пропускной способностью и время для завершения начальной синхронизации. Можно определить конфигурацию Bitcoin Core таким образом, чтобы сократить размер структуры данных блокчейна за счет отбрасывания старых блоков (см. пример 3.2), но для этого все равно потребуется загрузка всего объема данных, прежде чем появится возможность удаления данных.

Несмотря на жесткие требования к ресурсам, тысячи добровольных участников поддерживают работу биткойн-узлов. Некоторые узлы функционируют на таких простейших системах, как Raspberry Pi (компьютер[1] стоимостью 35 долларов и размером не более колоды карт). Многие добровольцы запускают биткойн-узлы на арендуемых серверах, обычно под управлением одного из вариантов ОС Linux. Кроме того, для запуска биткойн-узла можно использовать экземпляр виртуального выделенного сервера (Virtual Private Server, VPS) или сервера облачных вычислений (Cloud Computing Server). Многие провайдеры предлагают в аренду такие серверы с оплатой от 25 до 50 долларов в месяц.

Зачем вам нужен активный биткойн-узел? Ниже перечислены наиболее убедительные ответы на этот вопрос:

- если вы разработчик программного обеспечения биткойна, то вам необходим биткойн-узел для программируемого (API) доступа к сети и к структуре данных блокчейна;
- если вы создаете приложения, которые обязаны проверять корректность и допустимость (валидность) транзакций на соответствие правилам консенсуса в биткойн-системе. Обычно все компании, занимающиеся производством ПО для биткойна, имеют активные узлы;
- если вы хотите заниматься поддержкой биткойна. Создание новых узлов способствует повышению надежности и устойчивости биткойн-сети,

---

[1] Википедия уточняет: «В стандартный комплект поставки входит только сама плата. Корпус, блок питания, флэш-карту необходимо заказывать отдельно» (https://ru.wikipedia.org/wiki/Raspberry_Pi).

позволяет обслуживать больше кошельков и пользователей и выполнять больше транзакций;

- если нежелательно доверяться какой-либо третьей стороне при обработке или проверке (валидации) ваших транзакций.

Если вы читаете эту книгу и серьезно заинтересованы разработкой ПО для биткойна, то вам следует задуматься о необходимости запуска собственного узла.

Самый первый запуск Bitcoin Core

При первом запуске bitcoind будет выведено напоминание о необходимости создания файла конфигурации с надежным паролем для интерфейса JSON-RPC. Этот пароль управляет доступом к прикладному программному интерфейсу (API), предлагаемому Bitcoin Core.

В терминале выполните команду запуска `bitcoind`:

```
$ bitcoind
Error: To use the "-server" option, you must set a rpcpassword in the configuration file:
(Ошибка: для использования ключа "-server" необходимо установить пароль rpcpassword в файле
конфигурации:)
/home/ubuntu/.bitcoin/bitcoin.conf
It is recommended you use the following random password:
(Рекомендуется использовать приведенный ниже случайно выбранный пароль:)
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(you do not need to remember this password)
(этот пароль не обязательно запоминать)
The username and password MUST NOT be the same.
(Имя пользователя и пароль НЕ ДОЛЖНЫ быть одинаковыми.)
If the file does not exist, create it with owner-readable-only file permissions.
(Если файл не существует, то создайте его с ограничением доступа только для чтения владельцем.)
It is also recommended to set alertnotify so you are notified of problems;
(Также рекомендуется настроить механизм предупреждений, чтобы получать сообщения о возникших
проблемах; например:)
for example: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

Можно видеть, что при первом запуске `bitcoind` сообщает о необходимости создания файла конфигурации, в котором как минимум должны содержаться записи о пользователе `rpcuser` и его пароле `rpcpassword`. Кроме того, рекомендуется настроить механизм предупреждений. В следующем разделе мы более подробно рассмотрим различные параметры и опции конфигурации, а также пример файла конфигурации.

Конфигурирование узла Bitcoin Core

Отредактируйте файл конфигурации с помощью наиболее удобного для вас редактора текста и установите требуемые параметры, заменив пароль на более надежный, как рекомендует `bitcoind`. Не используйте пароли, приведенные в этой книге. Создайте файл конфигурации в подкаталоге *.bitcoin* вашего до-

машнего каталога, таким образом, это будет файл *.bitcoin/bitcoin.conf*, и определите в нем имя пользователя и пароль:

```
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

В дополнение к параметрам rpcuser и rpcpassword Bitcoin Core предлагает более 100 параметров конфигурации, регулирующих поведение узла сети, хранение структуры данных блокчейна и многие другие аспекты функционирования. Чтобы получить полный список этих параметров, выполните следующую команду:

```
$ bitcoind --help
Bitcoin Core Daemon version v0.11.2

Usage:
  bitcoind [options]                     Start Bitcoin Core Daemon (Запуск демона Bitcoin Core)

Options:

  -?
       This help message
       (вывод этого сообщения)

  -alerts
       Receive and display P2P network alerts (default: 1)
       (Получение и вывод предупреждений от пиринговой сети (по умолчанию: 1))

  -alertnotify=<cmd>
       Execute command when a relevant alert is received or we see a really
       long fork (%s in cmd is replaced by message)
       (Выполнить команду при получении соответствующего предупреждения или если наблюдается
       слишком длительная задержка при создании нового процесса (fork)
       (%s в команде cmd заменяется полученным сообщением))
...
[длинный список параметров]
...
  -rpcsslciphers=<ciphers>
       Acceptable ciphers (default:
       TLSv1.2+HIGH:TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!3DES:@STRENGTH)
```

Ниже приведены только самые важные параметры, которые вы можете установить в файле конфигурации или как параметры командной строки при запуске bitcoind:

`alertnotify`
: Позволяет запустить заданную команду или скрипт для отправки предупреждений владельцу узла, обычно по электронной почте.

`conf`
: Альтернативное местоположение файла конфигурации. Применение этого параметра имеет смысл только в качестве параметра командной строки для bitcoind, так как он не может находиться в файле конфигурации, на который сам же и ссылается.

`datadir`
: Определение каталога и файловой системы, в которой будут размещены все данные блокчейна. По умолчанию это подкаталог .bitcoin в вашем домашнем каталоге. Убедитесь в том, что в указанной файловой системе имеется несколько гигабайтов свободного пространства.

`prune`
: Позволяет несколько снизить требования к объему дискового пространства путем удаления старых блоков. Используйте этот параметр на узле с ограниченными ресурсами, недостаточными для поддержки полной структуры данных блокчейна.

`txindex`
: Сопровождение индекса всех транзакций. Это означает поддержку полной копии блокчейна, что позволяет программно получать доступ к любой транзакции по ее идентификатору.

`maxconnections`
: Установка максимального количества узлов, с которыми возможно установление соединений. Уменьшение значения, по сравнению с принятым по умолчанию, снизит нагрузку по пропускной способности вашего узла. Используйте этот параметр, если ваше соединение с Интернетом ограничено по объему данных или установлена оплата за каждый гигабайт.

`maxmempool`
: Ограничение размера пула памяти. Используйте этот параметр для уменьшения объема памяти, используемой узлом.

`maxreceivebuffer/maxsendbuffer`
: Ограничение размера буфера памяти для каждого соединения. Определяется в единицах, кратных 1000 байтов. Используется на узлах с ограниченным объемом памяти.

`minrelaytxfee`
: Установка минимальной суммы сбора за каждую передаваемую транзакцию. Если сумма меньше значения этого параметра, то считается, что транзакция передается с нулевой оплатой. Используйте этот параметр на узлах с ограниченным объемом памяти, чтобы уменьшить размер пула транзакций, выполняемых в оперативной памяти.

Индекс базы данных транзакций и параметр txindex

По умолчанию Bitcoin Core создает базу данных, содержащую только транзакции, связанные с кошельком пользователя. Если нужна возможность доступа к любой транзакции с помощью команд, подобных `getrawtransaction` (см. раздел «Обработка и расшифровка транзакций» ниже), то необходимо определить конфигурацию Bitcoin Core для создания полного индекса транзакций, что можно сделать с по-

мощью параметра txindex. Установите значение txindex=1 в файле конфигурации Bitcoin Core. Если этот параметр не был установлен перед первым запуском, а изменен позже, то потребуется перезапуск демона bitcoind с обязательным ключом -reindex, после чего придется подождать некоторое время, пока индекс не будет перестроен.

В примере 3.1 показано, как можно комбинировать описанные выше параметры для узла с полным индексированием, запускаемым в качестве внутреннего API-компонента биткойн-приложения.

Пример 3.1 ❖ Пример конфигурации узла с полным индексированием

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

В примере 3.2 показана конфигурация узла с ограниченными ресурсами, запускаемого на более слабом сервере.

Пример 3.2 ❖ Пример конфигурации системы с ограниченными ресурсами

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
minrelaytxfee=0.0001
maxmempool=200
maxreceivebuffer=2500
maxsendbuffer=500
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

После редактирования файла конфигурации и установки значений параметров, наилучшим образом соответствующих вашим требованиям, можно проверить функционирование bitcoind с заданной конфигурацией. Для этого нужно запустить Bitcoin Core с ключом -printtoconsole, обеспечивающим работу в активном режиме с выводом в консоль:

```
$ bitcoind -printtoconsole

Bitcoin version v0.11.20.0
Using OpenSSL version OpenSSL 1.0.2e 3 Dec 2015
Startup time: 2015-01-02 19:56:17
Using data directory /tmp/bitcoin
Using config file /tmp/bitcoin/bitcoin.conf
Using at most 125 connections (275 file descriptors available)
Using 2 threads for script verification
scheduler thread start
HTTP: creating work queue of depth 16
```

```
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /tmp/bitcoin/.cookie
HTTP: starting 4 worker threads
Bound to [::]:8333
Bound to 0.0.0.0:8333
Cache configuration:
* Using 2.0MiB for block index database
* Using 32.5MiB for chain state database
* Using 65.5MiB for in-memory UTXO set
init message: Loading block index...
Opening LevelDB in /tmp/bitcoin/blocks/index
Opened LevelDB successfully

[... большое количество сообщений ...]
```

Чтобы прервать выполнение активного процесса, можно нажать комбинацию клавиш **Ctrl+C**, если вы убедились в правильности значений всех параметров и программа работает так, как предполагалось.

Для запуска Bitcoin Core в фоновом режиме (это его основной режим работы) запустите его с ключом `-daemon`, то есть выполните команду `bitcoind -daemon`.

Для наблюдения за работой и текущим состоянием биткойн-узла воспользуйтесь следующей командой:

```
$ bitcoin-cli getinfo

{
    "version" : 110200,
    "protocolversion" : 70002,
    "blocks" : 396328,
    "timeoffset" : 0,
    "connections" : 15,
    "proxy" : "",
    "difficulty" : 120033340651.23696899,
    "testnet" : false,
    "relayfee" : 0.00010000,
    "errors" : ""
}
```

Здесь можно видеть, что на узле работает Bitcoin Core версии 0.11.2, высота структуры данных блокчейна составляет 396 328 блоков и установлено 15 активных сетевых соединений.

Если вы вполне удовлетворены выбранными значениями параметров конфигурации, то следует добавить команду запуска биткойн-системы в скрипты автозапуска вашей операционной системы, чтобы программа Bitcoin Core работала постоянно и автоматически перезапускалась при каждой перезагрузке операционной системы. Многочисленные примеры скриптов автозапуска для различных операционных систем можно найти в подкаталоге исходных кодов биткойна *contrib/init* и в файле *README.md*, где объясняется, какой скрипт для какой ОС предназначен.

Прикладной программный интерфейс (API) Bitcoin Core

В клиенте Bitcoin Core реализован интерфейс JSON-RPC, к которому также можно получить доступ с помощью вспомогательной утилиты командной строки bitcoin-cli. Командная строка позволяет экспериментировать в интерактивном режиме с функциональными возможностями, доступными в том числе и через прикладной программный интерфейс (API). Сначала выполните команду help, чтобы ознакомиться со списком доступных RPC-команд биткойна:

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] ( "account" )
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
decoderawtransaction "hexstring"
...
...
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

Каждая из этих команд может принимать несколько параметров. Чтобы получить расширенную подсказку с подробным описанием и информацией о параметрах, добавьте имя конкретной команды после help. Например, для вывода подробного описания RPC-команды getblockhash выполните следующую команду:

```
$ bitcoin-cli help getblockhash
getblockhash index

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

Arguments:
(Аргументы:)
1. index         (numeric, required) The block index
                 (число, обязательный)(Индекс блока)

Result:
(Результат:)
"hash"         (string) The block hash
               (строка) (Хэш-значение 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 выведенного описания можно видеть два примера применения этой RPC-команды с использованием вспомогательной утилиты `bitcoin-cli` или HTTP-клиента `curl`. Примеры показывают, как можно вызвать команду. Если скопировать первый пример и выполнить его, то получим следующий результат:

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

Результат представляет собой хэш-значение блока, которое более подробно будет рассматриваться в следующих главах. Сейчас просто отметим, что эта команда должна вывести точно такой же результат в вашей системе, подтверждая, что узел Bitcoin Core работает корректно, правильно выполняет команды и выдает достоверную информацию о блоке с номером 1000.

В следующих разделах мы более подробно рассмотрим некоторые полезные RPC-команды и выводимые ими результаты выполнения.

Получение информации о состоянии клиента Bitcoin Core

Команда: `getinfo`.

RPC-команда биткойна `getinfo` выводит важную информацию о состоянии узла биткойн-сети, кошелька и базы данных блокчейна. Для выполнения команды используйте вспомогательную утилиту `bitcoin-cli`:

```
$ bitcoin-cli getinfo
{
    "version" : 110200,
    "protocolversion" : 70002,
    "blocks" : 396367,
    "timeoffset" : 0,
    "connections" : 15,
    "proxy" : "",
    "difficulty" : 120033340651.23696899,
    "testnet" : false,
    "relayfee" : 0.00010000,
    "errors" : ""
}
```

Данные возвращаются в формате JavaScript Object Notation (JSON), который без затруднений обрабатывается практически всеми языками программирования, к тому же вполне удобен для чтения человеком. В представленных данных указаны номера версий программного клиента биткойна (110200) и протокола биткойна (70002). Также можно видеть высоту текущего блока, показывающую, сколько блоков известно этому клиенту (396367). Кроме того, показаны разнообразные статистические данные о биткойн-сети и некоторые параметры настройки текущего клиента.

Потребуется некоторое время, возможно, даже больше суток, для того чтобы клиент `bitcoind` определил текущую высоту структуры данных блокчейна, потому что блоки загружаются из других клиентов биткойна. Ход этого процесса можно проверять, используя команду `getinfo` для наблюдения за количеством известных блоков.

Обработка и расшифровка транзакций

Команды: `getrawtransaction`, `decoderawtransaction`.

В разделе «Покупка чашки кофе» главы 2 Алиса купила чашку кофе в кафе Боба. Ее транзакция была записана в структуру данных блокчейна с идентификатором txid 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2. Воспользуемся возможностями прикладного программного интерфейса (API) для извлечения и исследования этой транзакции, указав ее идентификатор как параметр для следующей команды:

```
$ bitcoin-cli getrawtransaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a↵
5fbd8a57286c345c2f2

0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa35779000↵
000008b483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4↵
ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813014↵
10484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc54123363767↵
89d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e3160000000↵
0001976a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000000000001976a9↵
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

Идентификатор транзакции не является полностью достоверным и заслуживающим доверия элементом данных до тех пор, пока транзакция не будет подтверждена. Отсутствие хэш-значения транзакции в структуре данных блокчейна не означает, что транзакция не была обработана. Это состояние известно под названием «деформативность транзакции» (transaction malleability), потому что транзакция может быть изменена до момента ее подтверждения в блоке. После подтверждения идентификатор `txid` становится неизменяемым и полностью достоверным.

Команда `getrawtransaction` возвращает сериализованные данные транзакции в шестнадцатеричной кодировке. Для ее расшифровки используется команда `decoderawtransaction`, в которую как параметр передаются данные в шестнадцатеричном формате. Можно скопировать шестнадцатеричные данные из вывода команды `getrawtransaction` и вставить их как параметр команды `decoderawtransaction`:

```
$ bitcoin-cli decoderawtransaction 0100000001186f9f998a5aa6f048e51dd8419a14d8↵
a0f1a8a2836dd734d2804fe65fa35779000000008b483045022100884d142d86652a3f47ba474↵
6ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298↵
cad530a863ea8f53982c09db8f6e381301410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fd↵
e0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa↵
336a8d752adfffffffff0260e31600000000001976a914ab68025513c3dbd2f7b92a94e0581f5↵
d50f654e788acd0ef8000000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8↵
88ac00000000

{
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
  "size": 258,
  "version": 1,
  "locktime": 0,
  "vin": [
```

```
    {
      "txid": "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig": {
        "asm":"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1decc...",
        "hex":"483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1de..."
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ab68...5f654e7 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
        ]
      }
    },
    {
      "value": 0.08450000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 7f9b1a...025a8 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
  ]
}
```

Расшифрованная транзакция показывает все свои компоненты, включая входные и выходные данные. В этом примере мы видим, что транзакция записала на кредит нашего нового адреса 50 миллибит, используя один фрагмент входных данных, и сгенерировала два фрагмента выходных данных. Входными данными для рассматриваемой транзакции стали выходные данные предыдущей подтвержденной транзакции (показанные как `vin` с `txid`, начинающимся с `7957a35fe`). Два фрагмента выходных данных соответствуют кредиту в 50 миллибит и сдаче, возвращаемой отправителю.

Можно продолжить исследование структуры данных блокчейна и рассмотреть предыдущую транзакцию, обозначенную собственным идентификато-

ром txid, используя те же команды (например, getrawtransaction). Переходя от транзакции к транзакции, мы можем проследить цепочку транзакций в обратном направлении, трактуя их как денежные средства, передаваемые от одного адреса владельца к новому адресу владельца.

Исследование блоков

Команды: getblock, getblockhash.

Исследование блоков похоже на исследование транзакций. Но к блокам можно обращаться с помощью высоты (height) блока или с помощью его хэш-значения (hash). Сначала найдем блок по его высоте. В разделе «Покупка чашки кофе» главы 2 было отмечено, что транзакция Алисы включена в блок 277316.

Воспользуемся командой getblockhash, которая в качестве параметра принимает высоту блока, а возвращает хэш-значение для этого блока:

```
$ bitcoin-cli getblockhash 277316
000000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Теперь мы знаем, в какой блок включена транзакция Алисы, и можем обратиться к этому блоку. Для этого используем команду getblock с хэш-значением блока, заданным как параметр:

```
$ bitcoin-cli getblock 000000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b3↵
1b2cc7bdc4

{
  "hash": "000000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
  "confirmations": 37371,
  "size": 218629,
  "height": 277316,
  "version": 2,
  "merkleroot":
"c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
  "tx": [
    "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",
    "04905ff987ddd4cfe603b03cfb7ca50ee81d89d1f8f5f265c38f763eea4a21fd",
    "32467aab5d04f51940075055c2f20bbd1195727c961431bf0aff8443f9710f81",
    "561c5216944e21fa29dd12aaa1a45e3397f9c0d888359cb05e1f79fe73da37bd",
[... сотни транзакций ...]
    "78b300b2a1d2d9449b58db7bc71c3884d6e0579617e0da4991b9734cef7ab23a",
    "6c87130ec283ab4c2c493b190c20de4b28ff3caf72d16ffa1ce3e96f2069aca9",
    "6f423dbc3636ef193fd8898dfdf7621dcade1bbe509e963ffbff91f696d81a62",
    "802ba8b2adabc5796a9471f25b02ae6aeee2439c679a5c33c4bbcee97e081196",
    "eaaf6a048588d9ad4d1c092539bd571dd8af30635c152a3b0e8b611e67d1a1af",
    "e67abc6bd5e2cac169821afc51b207127f42b92a841e976f9b752157879ba8bd",
    "d38985a6a1bfd35037cb7776b2dc86797abbb7a06630f5d03df2785d50d5a2ac",
    "45ea0a3f6016d2bb90ab92c34a7aac9767671a8a84b9bcce6c019e60197c134b",
    "c098445d748ced5f178ef2ff96f2758cbec9eb32cb0fc65db313bcac1d3bc98f"  ],
  "time": 1388185914,
```

```
  "mediantime": 1388183675,
  "nonce": 924591752,
  "bits": "1903a30c",
  "difficulty": 1180923195.258026,
  "chainwork":
"0000000000000000000000000000000000000000000000000934695e92aaf53afa1a",
  "previousblockhash":
"0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569",
  "nextblockhash":
"000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba7"
}
```

Блок содержит 419 транзакций, и 64-м по списку (0627052b...) является платеж Алисы за чашку кофе. Элемент height сообщает, что это блок с номером 277316 в структуре данных блокчейна.

Использование программного интерфейса Bitcoin Core

Вспомогательная утилита bitcoin-cli очень полезна при изучении прикладного программного интерфейса Bitcoin Core API и при тестировании функций. Но самая главная цель прикладного программного интерфейса – обеспечение программного доступа к функциям Bitcoin Core. В этом разделе будет показана организация доступа к Bitcoin Core из другой программы.

Bitcoin Core API – это интерфейс JSON-RPC. JSON означает JavaScript Object Notation и представляет собой очень удобный способ представления данных, которые без затруднений могут быть прочитаны как человеком, так и программами. RPC означает Remote Procedure Call (вызов удаленных процедур), то есть вызов процедур (функций), расположенных на дистанционно удаленном устройстве (на узле Bitcoin Core), с использованием сетевого протокола. В данном случае сетевым протоколом является HTTP или HTTPS (для защищенных соединений с шифрованием трафика).

Когда мы используем команду bitcoin-cli для вывода справочной информации о какой-либо команде, то выводится и пример применения curl, универсального HTTP-клиента для командной строки, для формирования одного из вариантов вызова JSON-RPC:

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

В этой команде показано, что curl выполняет HTTP-запрос к локальному хосту (127.0.0.1), устанавливает соединение с принятым по умолчанию портом (8332) для биткойн-системы и инициализирует запрос jsonrpc на выполнение метода getinfo с использованием кодировки типа text/plain.

Если реализация вызова JSON-RPC выполняется в создаваемой вами программе, то можно воспользоваться общецелевой HTTP-библиотекой для формирования вызова, похожего на вызов, показанный в предыдущем примере с утилитой curl.

Тем не менее практически для каждого языка программирования существуют специализированные библиотеки, обеспечивающие удобную «обертку» Bitcoin Core API и намного упрощающие работу с этим интерфейсом. Для упрощения доступа к API мы будем использовать библиотеку `python-bitcoinlib`. Но следует помнить, что вам потребуется работающий экземпляр Bitcoin Core, чтобы с его помощью выполнять вызовы JSON-RPC.

Скрипт на языке Python в примере 3.3 создает простой вызов метода `getinfo` и выводит значение параметра `blocks`, взятое из данных, возвращаемых Bitcoin Core.

Пример 3.3 ❖ Удаленный вызов метода getinfo через Bitcoin Core JSON-RPC API

```
from bitcoin.rpc import RawProxy

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

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

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

Запуск этого скрипта дает следующий результат:

```
$ python rpc_example.py
394075
```

Это говорит о том, что на нашем локальном узле Bitcoin Core содержится 394 075 блоков в соответствующей структуре данных блокчейна. Результат не очень впечатляющий, но этот пример демонстрирует основы применения выбранной библиотеки как упрощенного интерфейса к Bitcoin Core JSON-RPC API.

Далее используем вызовы функций `getrawtransaction` и `decodetransaction` для получения подробностей оплаты чашки кофе Алисой. В примере 3.4 транзакция Алисы извлекается из блока и выводится список выходных данных этой транзакции. Для каждого элемента выходных данных указываются адрес получателя и значение. Как уже было сказано ранее, транзакция Алисы состоит из фрагмента выходных данных с платежом кафе Боба и фрагмента со сдачей, возвращаемой Алисе.

Пример 3.4 ❖ Извлечение транзакции и обработка ее выходных данных

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# Идентификатор ID транзакции Алисы
txid = "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2"

# 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']['addresses'], output['value'])
```

Выполнив этот код, получим:

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA'], Decimal('0.01500000'))
([u'1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK'], Decimal('0.08450000'))
```

Оба приведенных выше примера достаточно просты. В действительности совсем не обязательно было писать программу для получения этих результатов, достаточно было бы просто воспользоваться вспомогательной утилитой bitcoin-cli. Но в следующем примере потребуется несколько сотен RPC-вызовов, и он более наглядно демонстрирует практическое применение программного интерфейса.

В примере 3.5 сначала извлекается блок 277316, затем – каждая из 419 транзакций с соответствующими идентификаторами. Далее выполняется последовательный проход по всем элементам выходных данных каждой транзакции и суммируются их значения.

Пример 3.5 ❖ Обработка блока и суммирование выходных данных всех транзакций

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# Высота блока, в котором записана транзакция Алисы
blockheight = 277316

# Получение хэш-значения блока, высота которого равна 277316
blockhash = p.getblockhash(blockheight)

# Извлечение блока по его хэш-значению
block = p.getblock(blockhash)

# Элемент tx содержит список идентификаторов всех транзакций в этом блоке
transactions = block['tx']

block_value = 0

# Итеративный проход по всем идентификаторам транзакций в блоке
for txid in transactions:
    tx_value = 0
    # Извлечение сырой транзакции по ее идентификатору
    raw_tx = p.getrawtransaction(txid)
    # Расшифровка транзакции
    decoded_tx = p.decoderawtransaction(raw_tx)
    # Итеративный проход по всем фрагментам выходных данных обрабатываемой транзакции
    for output in decoded_tx['vout']:
        # Добавление значения каждого фрагмента выходных данных к общей сумме транзакции
        tx_value = tx_value + output['value']
    # Добавление суммарного значения текущей транзакции к общей сумме всего блока
    block_value = block_value + tx_value

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

Результат выполнения этого кода будет следующим:

```
$ python rpc_block.py

('Total value in block: ', Decimal('10322.07722534'))
```

Код приведенного выше примера позволяет вычислить общую сумму, переданную в исследуемом блоке, которая равна 10322.07722534 BTC (включая 25 BTC вознаграждения и 0.0909 BTC транзакционных сборов). Сравните это значение с суммой, полученной на сайте проводника блокчейна в результате поиска по хэш-значению или по высоте того же блока. Некоторые проводники сообщают суммарное значение без учета вознаграждения и транзакционных сборов. Различие в суммах легко объяснимо и не должно вызвать затруднений.

Прочие клиенты, библиотеки и инструментальные пакеты

Существует множество разнообразных клиентов, библиотек, инструментальных пакетов и даже полноценных реализаций узлов экосистемы биткойна. Они написаны на различных языках программирования и предлагают разработчикам интерфейсы, специализированные для конкретных языков программирования, предпочитаемых тем или иным программистом.

В следующих разделах приводятся списки с кратким описанием наиболее известных библиотек, клиентов и инструментальных пакетов с классификацией по языкам программирования.

C/C++

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

libbitcoin (https://github.com/libbitcoin/libbitcoin)
Кросс-платформенная библиотека инструментальных средств разработки узла и поддержки механизма консенсуса на языке C++.

bitcoin explorer (https://github.com/libbitcoin/libbitcoin-explorer)
Инструмент с интерфейсом командной строки для библиотеки libbitcoin.

picocoin (https://github.com/jgarzik/picocoin)
Упрощенная библиотека поддержки клиента на языке C, автор Джефф Гарзик (Jeff Garzik).

JavaScript

bcoin (http://bcoin.io)
Хорошо организованная (по модулям) масштабируемая реализация полноценного узла с прикладным программным интерфейсом (API).

Bitcore (https://bitcore.io/)
Полноценный узел API-интерфейса и библиотека от Bitpay.

BitcoinJS (https://github.com/bitcoinjs/bitcoinjs-lib)
Написанная исключительно на JavaScript библиотека Bitcoin, специализированная для node.js и браузеров.

Java

bitcoinj (https://bitcoinj.github.io)
Библиотека поддержки клиента для полноценного узла, написанная исключительно на Java.

Bits of Proof (BOP) (https://bitsofproof.com)
Реализация биткойн-системы на уровне предприятия (enterprise-class) на языке Java.

Python

python-bitcoinlib (https://github.com/petertodd/python-bitcoinlib)
Библиотека для создания биткойн-системы, библиотека поддержки механизма консенсуса и создания узла на языке Python, автор Питер Тодд (Peter Todd).

pycoin (https://github.com/richardkiss/pycoin)
Библиотека поддержки биткойна, автор Ричард Кисс (Richard Kiss).

pybitcointools (https://github.com/vbuterin/pybitcointools)
Библиотека поддержки биткойна, автор Виталик Бутерин (Vitalik Buterin).

Ruby

bitcoin-client (https://github.com/sinisterchipmunk/bitcoin-client)
Библиотека функций-оберток на языке Ruby для JSON-RPC API.

Go

btcd (https://github.com/btcsuite/btcd)
Биткойн-клиент с реализацией полноценного узла на языке Go.

Rust

rust-bitcoin (https://github.com/apoelstra/rust-bitcoin)
Библиотека поддержки биткойна на языке Rust с функциями сериализации, синтаксического разбора и API-вызовами.

C#

NBitcoin (https://github.com/MetacoSA/NBitcoin)
Библиотека полноценной поддержки биткойн-системы для программной среды .NET.

Objective-C

CoreBitcoin (https://github.com/oleganza/CoreBitcoin)
Инструментальный пакет поддержки биткойна для Objective-C и Swift.

Кроме того, существует множество разнообразных библиотек для других языков программирования, и постоянно создаются новые библиотеки и инструментальные средства.

Глава 4

Ключи и адреса

Возможно, вам известно, что биткойн-система основана на криптографии (cryptography), являющейся одним из разделов математики и интенсивно используемой в области обеспечения компьютерной безопасности. По-гречески криптография означает «секретное письмо», но как наука криптография заключает в себе больше, чем простое написание секретных сообщений, которое обозначается термином «зашифрование» (encryption). Криптография также может применяться для доказательства знания секретной информации без раскрытия самой информации (цифровая подпись; digital signature) или для подтверждения достоверности данных (цифровой отпечаток; digital fingerprint). Эти типы криптографических доказательств являются математическими инструментами, крайне важными для биткойна и активно используемыми в биткойн-приложениях. По иронии судьбы как раз зашифрование не является столь важной частью биткойн-системы, так как передаваемые сообщения и данные транзакций не шифруются, поскольку в этом нет необходимости при защите денежных средств. В этой главе дается краткое введение в криптографию, точнее в ту ее часть, которая используется в биткойн-системах для управления правами владения денежными средствами в форме ключей, адресов и кошельков.

Введение

Право владения биткойнами устанавливается с помощью цифровых ключей (digital keys), биткойн-адресов (bitcoin addresses) и цифровых подписей (digital signatures). Цифровые ключи в действительности не хранятся в сети, а создаются и сохраняются пользователями в файле или в простой базе данных, называемой кошельком (wallet). Цифровые ключи в кошельке пользователя абсолютно независимы от протокола биткойна, поэтому могут генерироваться и управляться программным обеспечением пользовательского кошелька без обращений к структуре данных блокчейна и без доступа к Интернету. Ключи создают много интересных свойств биткойна, включая распределенные доверительные отношения и распределенное управление, засвидетельствование права владения и модель защиты, основанную на криптографическом доказательстве.

Большинство транзакций биткойнов требует корректной цифровой подписи, обязательно включаемой в структуру данных блокчейна. Корректная цифровая подпись может быть сгенерирована только с помощью секретного ключа, следовательно, обладатель копии этого ключа управляет биткойнами. Цифровая подпись применяется при расходовании денежных средств и также обозначается термином «свидетельство» (witness), используемым в криптографии. Данные свидетельства в биткойн-транзакции подтверждают истинность права владения расходуемыми денежными средствами.

Ключи создаются парами, состоящими из закрытого (private) или секретного ключа (secret key) и открытого ключа (public key). Открытый ключ можно считать некоторым аналогом номера банковского счета, а секретный ключ – аналогом секретного личного идентификационного номера (PIN) или подписи на чеке, то есть средством управления счетом. Пользователи биткойн-системы крайне редко видят эти цифровые ключи. В основном ключи хранятся в файле кошелька и управляются программным обеспечением биткойн-кошелька.

В платежной части биткойн-транзакции открытый ключ получателя представлен его цифровым отпечатком, называемым биткойн-адресом (bitcoin address), который используется тем же способом, что и имя получателя денег по чеку (то есть «Выплатить предъявителю чека»). В большинстве случаев биткойн-адрес генерируется по открытому ключу и однозначно соответствует ему. Но не все биткойн-адреса представляют открытые ключи, они могут также представлять других получателей, таких как скрипты, как вы увидите несколько позже в этой главе. Используемые подобным способом биткойн-адреса отделяются от конкретного получателя денежных средств, что позволяет более гибко назначать цели транзакций, подобно тому, как это делается в бумажном чеке как едином платежном инструменте, применяемом для перевода денег на личные счета и счета компаний, для оплаты по счетам-фактурам или для выплаты наличными. Биткойн-адрес – это всего лишь форма представления ключей, с которыми пользователи встречаются регулярно, поскольку это часть той информации, которой необходимо поделиться с окружающим миром.

В начале главы будет представлено введение в криптографию и описаны математические методы, используемые в биткойн-системе. Затем будут рассматриваться генерация ключей, способы их хранения и управления ими. Далее следует краткое описание различных форматов кодирования, используемое для представления секретных и открытых ключей, адресов и адресов скриптов. В конце главы мы рассмотрим более сложные способы применения ключей и адресов: «престижный» (vanity) кошелек, мультиподписи, адреса скриптов и бумажные кошельки.

Криптография с открытым ключом и криптовалюта

Криптография с открытым ключом изобретена в 1970-х годах и представляет собой математическую основу для компьютерной и информационной безопасности.

После появления криптографии с открытым ключом были разработаны методики практического применения соответствующих математических функций, такие как возведение в степень простых чисел и умножение на эллиптических кривых. Эти математические функции практически необратимы, то есть вычисления в одном направлении достаточно просты, а в обратном направлении невыполнимы. Подобные математические функции как основа новой криптографии позволили создать цифровые методики надежного шифрования и цифровые подписи, которые невозможно подделать. Биткойн использует методику умножения на эллиптических кривых (умножения точек эллиптических кривых) как основную криптографическую методику.

В биткойн-системе криптография с открытым ключом применяется для создания пары ключей, управляющих доступом к биткойнам. Эта пара состоит из секретного ключа (private key) и производного от него уникального открытого ключа (public key). Открытый ключ используется для получения денежных средств, а секретный ключ – для подписи транзакций, расходующих денежные средства.

Между открытым и секретным ключами существует математическая зависимость, которая позволяет применять секретный ключ для генерации подписей к сообщениям. Достоверность такой подписи может быть проверена с помощью открытого ключа без раскрытия секретного ключа.

При расходовании биткойнов текущий владелец предоставляет свой открытый ключ и цифровую подпись (подписи различные для каждого случая, но создаются с помощью одного и того же секретного ключа) в транзакции, определяющей расход биткойнов. Благодаря наличию открытого ключа и цифровой подписи любой член биткойн-сети может проверить транзакцию и принять ее как корректную, подтвердив, что лицо, передающее биткойны, действительно является их владельцем в момент передачи.

В большинстве реализаций кошельков секретный и открытый ключи для удобства хранятся вместе как объединенная пара ключей. Но открытый ключ всегда можно вычислить по секретному ключу, поэтому также возможно хранение только одного секретного ключа.

Секретный ключ и открытый ключ

Кошелек биткойнов содержит набор пар ключей, каждая из которых состоит из секретного ключа и открытого ключа. Секретный ключ (k) – это число, обычно выбранное случайным образом. Методика умножения на эллиптических кривых (или умножения точек на эллиптических кривых), односторонняя криптографическая функция, используется для генерации открытого ключа (K) по секретному ключу. К полученному открытому ключу (K) применяется односторонняя криптографическая хэш-функция для генерации биткойн-адреса (A). Этот раздел начинается с описания генерации секретного ключа, затем рассматривается математика эллиптических кривых, используемая для получения открытого ключа, после чего объясняется генерация биткойн-адреса из открытого ключа. Взаимосвязи между секретным ключом, открытым ключом и биткойн-адресом показаны на рис. 4.1.

Рис. 4.1 ❖ Секретный ключ, открытый ключ и биткойн-адрес

Почему используется асимметричная критография (с открытыми/секретными ключами)?

Почему в биткойн-системе используется именно асимметричная криптография? Она не предназначена для «зашифрования» (сохранения в секрете) содержимого транзакций. Другим полезным свойством асимметричной криптографии является возможность генерации цифровых подписей (digital signatures). Секретный ключ можно применить к цифровым отпечаткам транзакции для получения подписи в числовой форме. Такая подпись может быть создана только тем, кому известен секретный ключ. Но каждый может получить доступ к открытому ключу, а цифровой отпечаток транзакции в сочетании с открытым ключом позволяет проверить достоверность подписи. Это полезнейшее свойство асимметричной криптографии предоставляет возможность проверить каждую подпись каждой транзакции при условии, что только владелец секретных ключей может создавать корректные подписи.

Секретные ключи

Секретный ключ (private key) – это просто число, выбранное случайным образом. Владение и управление секретным ключом является основой пользовательского управления всеми денежными средствами, связанными с соответствующим биткойн-адресом. Секретный ключ используется для создания подписей, необходимых для расходования биткойнов, чтобы подтвердить право владения денежными средствами, указанными в транзакции. Значение секретного ключа непременно нужно сохранять в тайне постоянно, так как знание его посторонним лицом равносильно передаче этому лицу управления всеми биткойнами, защищенными данным ключом. Кроме того, необходимо создавать резервные копии секретного ключа и защитить его от случайной потери, поскольку потерянный ключ восстановить невозможно, следовательно, безвозвратно теряются и денежные средства, защищенные этим ключом.

Секретный ключ для биткойнов – это просто число. Вы можете выбирать секретные ключи случайным образом с помощью монетки, карандаша и бумаги: подбросьте монетку 256 раз – и получите двоичные цифры случайного секретного ключа, который можно использовать в биткойн-кошельке. После этого из полученного секретного ключа можно сгенерировать открытый ключ.

Генерация секретного ключа из случайного числа

Первым и наиболее важным этапом процесса генерации ключей является поиск надежного источника энтропии, или случайности. Создание биткойн-ключа достаточно точно описывается высказыванием: «Выбрать число между 1 и 2^{256}». Конкретная методика выбора этого числа не имеет значения, так как число непредсказуемо и не повторяется. Программное обеспечение биткойна пользуется генераторами случайных чисел, предоставляемыми операционными системами, для получения случайного 256-битового числа. Обычно генератор случайных чисел ОС инициализируется некоторым произвольным значением, заданным человеком, вот почему, например, ОС может попросить вас подвигать мышь в течение нескольких секунд.

Если говорить более точно, секретный ключ может быть любым числом в диапазоне от 1 до n-1, где n – константа (n = 1.158 * 10^{77}, это чуть меньше, чем 2^{256}), определяемая как порядок эллиптической кривой, используемой в биткойн-системе (см. раздел «Криптография с использованием эллиптических кривых» ниже в этой главе). Для создания такого ключа случайно выбирается 256-битовое число и проверяется, является ли оно меньшим, чем n-1. С точки зрения программирования обычно это достигается предоставлением более длинной строки случайных битов, выбранных из криптографически надежного источника энтропии, в качестве входных данных для хэш-алгоритма SHA256, который гарантированно генерирует 256-битовое число. Если результат меньше n-1, то сгенерирован допустимый секретный ключ. В противном случае нужно просто повторить вычисление с другим случайным числом.

Не пишите свой код, создающий случайные числа, и не пользуйтесь «простыми» генераторами случайных чисел, предлагаемыми практически всеми языками программирования. Используйте криптографически безопасный генератор псевдослучайных чисел (CSPRNG) с затравочным ключом (seed) из источника достаточно сильной энтропии. Внимательно изучите документацию выбранной библиотеки генератора случайных чисел, чтобы полностью убедиться в ее криптографической надежности. Правильная реализация CSPRNG чрезвычайно важна для защиты ключей.

Ниже приводится случайно сгенерированный секретный ключ (k) в шестнадцатеричном формате (256 битов показаны в виде 64 шестнадцатеричных цифр, каждая из которых представляет 4 бита):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

Размер пространства секретных ключей биткойна (2^{256}) – действительно огромное число. В десятичном представлении это число приблизительно равно 10^{77}. Для сравнения: в наблюдаемой вселенной количество атомов по приблизительным оценкам равно 10^{80}.

Чтобы сгенерировать новый ключ с помощью клиента Bitcoin Core (см. главу 3), используйте команду `getnewaddress`. Из соображений безопасности она показывает только открытый ключ, но не секретный ключ. Для вывода секретного ключа утилитой `bitcoind` воспользуйтесь командой `dumpprivkey`. Эта команда

выводит секретный ключ в формате Base58 с проверкой контрольной суммы, называемом *Wallet Import Format (WIF)*, который более подробно рассматривается в разделе «Форматы секретного ключа» ниже в этой главе. Вот пример генерации и вывода секретного ключа с применением двух вышеупомянутых команд:

```
$ bitcoin-cli getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
$ bitcoin-cli dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Команда `dumpprivkey` открывает кошелек и извлекает из него секретный ключ, сгенерированный командой `getnewaddress`. Утилита `bitcoind` не имеет возможности узнать секретный ключ по открытому ключу, если оба этих ключа не хранятся в кошельке.

Команда `dumpprivkey` не генерирует секретный ключ из открытого ключа, это абсолютно невозможно. Эта команда просто предъявляет секретный ключ, который уже известен кошельку и сгенерирован ранее командой `getnewaddress`.

Вы также можете воспользоваться утилитой командной строки Bitcoin Explorer (см. приложение Ж) для генерации и вывода секретных ключей с помощью команд `seed`, `ec-new` и `ec-to-wif`:

```
$ bx seed | bx ec-new | bx ec-to-wif
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Открытые ключи

Открытый ключ (public key) вычисляется по секретному ключу с применением методики умножения на эллиптических кривых, которая является необратимой: K = k * G, где k – секретный ключ, G – постоянная точка эллиптической кривой, называемая базовой точкой генерации (generator point), K – вычисляемый открытый ключ. Обратная операция под названием «поиск дискретного логарифма» – вычисление k по известному значению K – является вычислением, трудоемкость которого сравнима с перебором всех известных значений k, то есть с методикой полного перебора (или «грубой силы»). Перед демонстрацией процедуры генерации открытого ключа по секретному ключу мы немного подробнее рассмотрим криптографию с использованием эллиптических кривых.

Умножение на эллиптической кривой (или умножение точек эллиптической кривой) – это тип функции, который криптографы называют «односторонней функцией с потайным входом» (trapdoor function): вычисление легко провести в одном направлении (умножение), но практически невозможно в обратном направлении (деление). Владелец секретного ключа может без труда создать открытый ключ и предъявить его всему миру, зная, что никто не сможет применить обратную функцию и вычислить секретный ключ по этому открытому ключу. Этот хитрый математический прием стал основой для надежно защищенных цифровых подписей, которые невозможно подделать и которые служат доказательством права владения биткойнами.

Криптография с использованием эллиптических кривых

Криптография с использованием эллиптических кривых – особый тип асимметричной криптографии или криптографии с открытым ключом, основанный на задаче дискретного логарифмирования, которую можно описать как сложение и умножение точек на эллиптической кривой.

На рис. 4.2 приведен пример эллиптической кривой, подобной используемой в биткойн-системе.

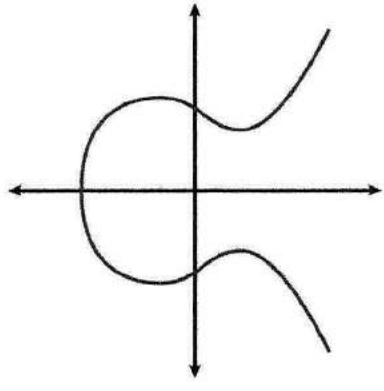


Рис. 4.2 ❖ Эллиптическая кривая

В биткойн-системах используются особенная эллиптическая кривая и набор математических констант, определенных в стандарте secp256k1, установленном Национальным институтом стандартов и технологий США (The National Institute of Standards and Technology, NIST). Кривая по стандарту secp256k1 определяется следующей функцией, которая формирует эллиптическую кривую:

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

или

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

Выражение *mod p* (взятие остатка при делении на простое число p) означает, что это кривая над конечным полем с характеристикой порядка простого числа p, также записываемой в виде $\mathbb{F}_p$, где $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ – очень большое простое число.

Поскольку эта кривая определена над конечным полем простого порядка, а не над полем действительных (или вещественных) чисел, она выглядит как набор точек, распределенных по двумерному пространству, что затрудняет ее визуальное представление. Но математические методы остаются теми же, что и для эллиптической кривой над полем действительных чисел. Пример на рис. 4.3 показывает аналогичную эллиптическую кривую над существенно меньшим конечным полем с характеристикой порядка простого числа 17

в виде набора точек на координатной сетке. Эллиптическую кривую биткойна по стандарту secp256k1 можно представить себе как значительно более сложный набор точек на неограниченной координатной сетке.

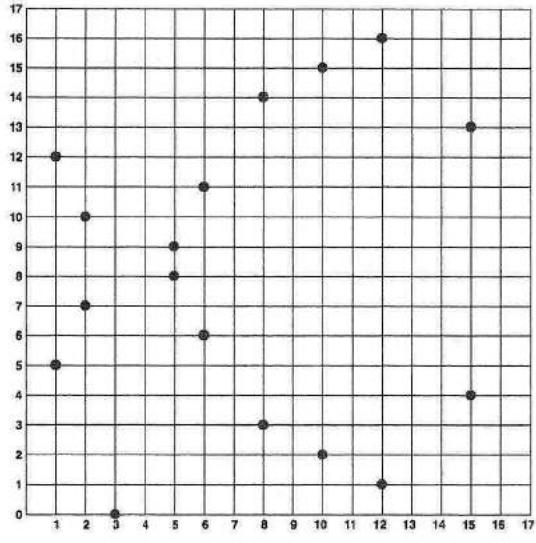


Рис. 4.3 ❖ Криптография с использованием эллиптических кривых: наглядное изображение эллиптической кривой над полем F(p) при p = 17

В качестве примера возьмем точку P с координатами (x,y), которая является одной из точек эллиптической кривой, соответствующей стандарту secp256k1:

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

В примере 4.1 показано, как можно проверить соответствие стандарту, используя язык программирования Python:

Пример 4.1 ❖ Использование ЯП Python для доказательства принадлежности точки стандартной эллиптической кривой

```
Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
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 существует так называемая «точка в бесконечности» (point at infinity), которая приближенно соответствует роли нуля в операции сложения. При компьютерных вычислениях это иногда представлено равенством $x = y = 0$ (не является решением уравнения эллиптической кривой, но позволяет сразу выделить случай, который легко проверить).

Также имеется оператор + (сложение), обладающий некоторыми свойствами, похожими на свойства обычной операции сложения действительных чисел, которую изучают в начальной школе. Если взять две точки P_1 и P_2 на эллиптической кривой, то существует третья точка $P_3 = P_1 + P_2$, также принадлежащая этой эллиптической кривой.

Геометрически эта третья точка P_3 определяется построением линии между точками P_1 и P_2. Линия пересекает эллиптическую кривую в одном и только в одном месте. Назовем эту точку пересечения $P'_3 = (x, y)$. Затем найдем ее зеркальное отображение относительно оси X, чтобы получить искомую точку $P_3 = (x, -y)$.

Кроме того, существует несколько особых случаев, которые объясняют необходимость точки в бесконечности.

Если точки P_1 и P_2 совпадают, то линия «между» ними должна представлять собой касательную к рассматриваемой кривой в точке P_1 (P_2). Эта касательная пересекает кривую в одной и только в одной новой точке. Можно воспользоваться методами математического анализа для определения угла наклона этой касательной линии. Эти методы работают несколько необычно, даже если ограничиться только теми точками на кривой, которые имеют обе целочисленные координаты.

В некоторых случаях (например, точки P_1 и P_2 имеют совпадающие координаты x, но различные координаты y) касательная будет строго вертикальной, а точка P_3 = «точка в бесконечности».

Если P_1 является точкой в бесконечности, то $P_1 + P_2 = P_2$. Аналогично, если P_2 является точкой в бесконечности, то $P_1 + P_2 = P_1$. Это показывает роль точки в бесконечности как нуля в операции сложения.

Операция сложения (+) является ассоциативной, то есть $(A + B) + C = A + (B + C)$. Это означает, что выражение $A + B + C$ можно записывать без скобок и при этом не возникает никакой неоднозначности.

После определения операции сложения и ее свойств можно определить операцию умножения стандартным способом как многократно повторяемую операцию сложения. Для любой точки P на эллиптической кривой, если k – целое число, то $kP = P + P + P + \ldots + P$ (k раз). Отметим, что в этом случае k иногда неправильно называют «экспонентой».

Генерация открытого ключа

Получив секретный ключ в форме случайным образом сгенерированного числа k, мы умножаем его на предварительно определенную точку эллиптической кривой, называемой базовой точкой генерации G (generator point G), для вы-

числения другой точки, также принадлежащей этой кривой, которая и является соответствующим открытым ключом K. Базовая точка генерации определена как часть стандарта secp256k1 и всегда является постоянной величиной для всех ключей биткойна:

$$K = k * G,$$

где k – секретный ключ, G – базовая точка генерации, K – вычисляемый открытый ключ, точка на эллиптической кривой. Так как базовая точка генерации всегда одинакова для всех пользователей биткойна, результатом умножения некоторого секретного ключа k на G всегда будет один и тот же открытый ключ K. Отношение между k и K жестко зафиксировано, но вычисление может быть выполнено только в одном направлении – от k к K. Именно поэтому биткойн-адрес (производный от K) можно сообщать всем, не открывая при этом пользовательский секретный ключ k.

Секретный ключ может быть преобразован в открытый ключ, но открытый ключ невозможно преобразовать обратно в секретный ключ, потому что описанный выше математический метод работает только в одном направлении.

Выполняя умножение точек на эллиптической кривой, мы берем секретный ключ k, сгененрированный заранее, и умножаем его на базовую точку генерации G, чтобы получить открытый ключ 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 операции умножения точки на целое число воспользуемся упрощенной эллиптической кривой над полем действительных чисел, поскольку ранее уже отмечалось, что для таких кривых математические методы остаются неизменными. Наша цель – найти произведение kG для базовой точки генерации G, или, что одно и то же, последовательно сложить точку G с самой собой k раз. На эллиптических кривых сложение точки с собой равносильно построению касательной в этой точке и определению новой точки пересечения касательной с кривой, после чего выполняется зеркальное отражение новой найденной точки пересечения относительно оси X.

На рис. 4.4 показан процесс получения значений G, 2G, 4G как последовательность геометрических построений на рассматриваемой кривой.

В большинстве реализаций биткойна для выполнения математических вычислений с эллиптическими кривыми используется библиотека OpenSSL cryptographic library (https://wiki.openssl.org/index.php/Elliptic_Curve_Cryptography). Например, для вычисления открытого ключа применяется функция EC_POINT_mul().

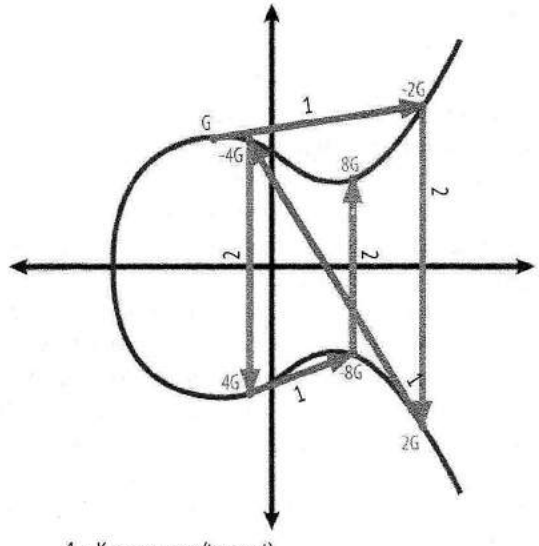


Рис. 4.4 ❖ Криптография с использованием эллиптических кривых: наглядное представление процедуры умножения точки G на целое число k на эллиптической кривой

Биткойн-адреса

Биткойн-адрес (bitcoin address) – это строка цифр и символов, которую можно сообщить каждому, кто намерен переслать вам деньги. Адреса, генерируемые на основе открытых ключей, представляют собой строку цифр и букв, всегда начинающуюся с цифры 1. Ниже приведен пример биткойн-адреса:

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

Биткойн-адрес чаще всего появляется в транзакциях как «получатель» денежных средств. Если сравнивать транзакцию биткойнов с бумажным чеком, то биткойн-адрес – это аналог получателя по чеку, то есть лица, указанного в строке после слов «Выплатить предъявителю:». В бумажном чеке в качестве получателя может быть указано имя владельца банковского счета, но, кроме того, получателем могут быть корпорации, организации, или даже может быть определена выплата наличными предъявителю чека. Поскольку в бумажном чеке не обязательно указывать номер счета, вместо этого используется произвольное имя получателя денежных средств, чеки являются в высшей степени универсальными платежными инструментами. Транзакции биткойнов используют похожую абстракцию – биткойн-адрес, что также делает их универсальными платежными инструментами. Биткойн-адрес может представлять владельца пары секретного/открытого ключа или что-либо другое, напри-

мер платежный скрипт, как мы увидим в разделе «Pay-to-Script-Hash (P2SH)» в главе 7. А сейчас рассмотрим простой случай – биткойн-адрес, представляющий открытый ключ, по которому он сгенерирован.

Биткойн-адрес генерируется на основе открытого ключа с использованием односторонней функции криптографического хэширования. Алгоритм хэширования, или просто хэш-алгоритм (hash algorithm), представляет собой одностороннюю (однонаправленную) функцию, которая вычисляет цифровой отпечаток (fingerprint) или хэш-значение (hash) входных данных произвольного размера. Криптографические хэш-функции активно используются в биткойн-системе: в биткойн-адресах, в адресах скриптов и в процессе майнинга по алгоритму доказательства выполнения работы (Proof-of-Work). К алгоритмам, применяемым для генерации биткойн-адреса по открытому ключу, относятся Secure Hash Algorithm (SHA) и RACE Integrity Primitives Evaluation Message Digest (RIPEMD), в частности версии SHA256 и RIPEMD160.

Имея открытый ключ K, мы вычисляем хэш-значение по алгоритму SHA256, а к полученному результату применяем алгоритм RIPEMD160, получая в итоге 160-битовое (20-байтовое) число:

A = RIPEMD160(SHA256(K)),

где K – открытый ключ, A – вычисляемый биткойн-адрес.

Биткойн-адрес – это не то же самое, что открытый ключ. Биткойн-адрес генерируется на основе открытого ключа с использованием односторонней (однонаправленной) функции.

Биткойн-адреса почти всегда кодируются в формате Base58Check (см. раздел «Форматы кодирования Base58 и Base58Check» ниже в этой главе), который использует 58 символов (система счисления Base59) и контрольную сумму для повышения удобства чтения человеком, для устранения неоднозначности и для защиты от ошибок в записи адреса и его ввода. Кроме того, формат Base58Check используется многими другими способами в биткойн-системе, там, где необходимо, чтобы пользователь без затруднений прочитал и правильно записал числовое значение, например биткойн-адрес, секретный ключ, зашифрованный ключ или хэш скрипта. В следующем разделе мы подробно рассмотрим механизм кодирования и декодирования Base58Check, а также представления результатов его работы. На рис. 4.5 показан процесс преобразования открытого ключа в биткойн-адрес.

Форматы кодирования Base58 и Base58Check

Для более компактного представления длинных чисел с использованием меньшего количества символов многие компьютерные системы используют смешанные алфавитно-цифровые представления с основанием (base или radix), большим 10. Например, в привычной для нас десятичной системе используются десять цифр от 0 до 9, а в шестнадцатеричной системе «цифр» 16 с шестью

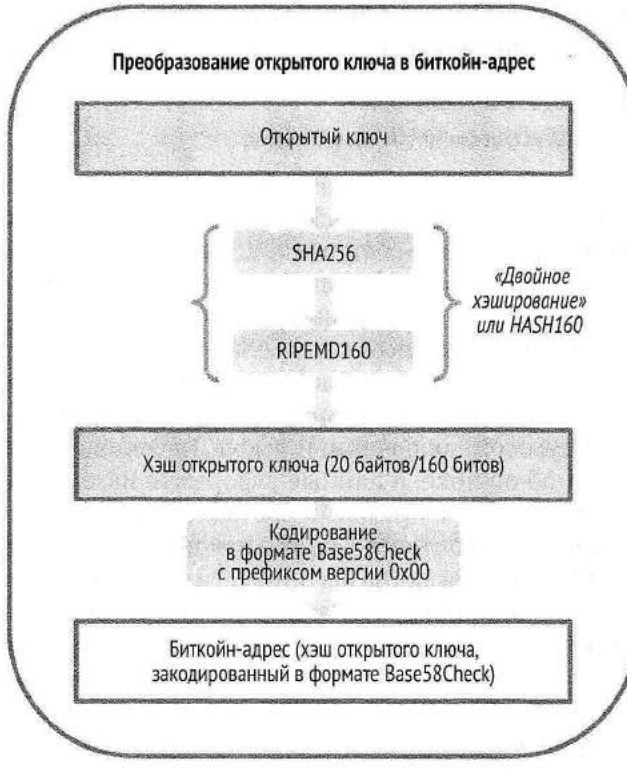


Рис. 4.5 ❖ Схема преобразования открытого ключа в биткойн-адрес

дополнительными символами – буквами от A до F. Число, представленное в шестнадцатеричном формате, короче, чем равное ему десятичное представление. Еще более компактное представление Base64 использует 26 прописных и 26 строчных букв латинского алфавита, 10 цифр и 2 дополнительных символа «+» (плюс) и «/» (слэш) для передачи бинарных данных с помощью средств связи, основанных на текстовом формате, таких как электронная почта. Чаще всего формат Base64 применяется для прикрепления бинарных вложений к сообщению электронной почты. Base58 – это основанный на тексте формат кодирования бинарных данных, разработанный для биткойн-систем и используемый многими другими криптовалютами. Он обеспечивает баланс между компактностью представления, удобством чтения и возможностью обнаружения и предотвращения ошибок. Base58 является подмножеством Base64, поскольку также использует заглавные и строчные буквы латинского алфавита и цифры, но исключает символы, которые часто путают друг с другом и которые могут выглядеть одинаково при отображении некоторыми шрифтами. Фактически Base58 – это Base64 без цифры 0 (нуль), без букв O (заглавная o), l (строчная L), I (заглавная i) и без символов + и /. Проще говоря, это набор из

заглавных и строчных букв и цифр без четырех вышеупомянутых (0, O, l, I). В примере 4.2 показан полный алфавит Base58.

Пример 4.2 ❖ Алфавит Base58 для биткойна

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

Для дополнительной защиты от опечаток и ошибок при записи в биткойн-системах часто используется Base58Check – формат кодирования Base58 с добавлением встроенной проверки на ошибки. Контрольная сумма (checksum) – это дополнительные 4 байта, добавляемые в конец закодированных данных. Контрольная сумма вычисляется по хэш-значению закодированных данных, следовательно, может использоваться для обнаружения и предотвращения ошибок и опечаток. При получении кода в формате Base58Check декодирующее программное обеспечение вычисляет контрольную сумму данных и сравнивает ее с контрольной суммой, включенной в код. Несовпадение контрольных сумм свидетельствует об ошибке, и данные в формате Base58Check считаются некорректными. Это защищает программное обеспечение кошелька от приема неправильно введенного биткойн-адреса как допустимого адреса получателя, то есть от ошибки, которая могла бы привести к потере денежных средств.

Для преобразования данных (числа) в формат Base58Check сначала к данным добавляется префикс, называемый байтом версии (version byte), который служит для простой идентификации типа кодируемых данных. Например, для биткойн-адреса префикс равен нулю (0x00 в шестнадцатеричном формате), а префикс при кодировании секретного ключа равен 128 (0x80 в шестнадцатеричном формате). Список наиболее часто используемых префиксов версий приведен в табл. 4.1.

Далее вычисляется двукратная SHA контрольная сумма, то есть к предыдущему результату (префикс и данные) два раза применяется хэш-алгоритм SHA256:

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

Из полученного 32-байтового хэш-значения (хэша от хэша) берутся только первые четыре байта, которые служат в качестве кода для проверки ошибок или контрольной суммы (checksum). Вычисленная контрольная сумма добавляется в конец данных.

Таким образом, конечный результат состоит из трех частей: префикс, собственно данные и контрольная сумма. Этот результат кодируется с использованием алфавита Base58, описанного выше. На рис. 4.6 показан процесс кодирования в формат Base58Check.

В биткойн-системе большинство данных предъявляется пользователю в кодировке Base58Check для компактности представления, удобства чтения и быстрого обнаружения ошибок. Префикс версии в кодировке Base58Check используется для создания легко различаемых форматов, которые при кодировании в Base58 содержат специальные символы в самом начале блока основных закодированных данных. Эти символы позволяют человеку без труда опреде-

лить тип закодированных данных и решить, как их использовать. Например, всегда можно отличить биткойн-адрес, который в кодировке Base58Check начинается с 1, от секретного ключа WIF, в кодировке Base58Check начинающегося с 5. Некоторые примеры префиксов версий и соответствующих им символов в итоговой кодировке Base58 показаны в табл. 4.1.

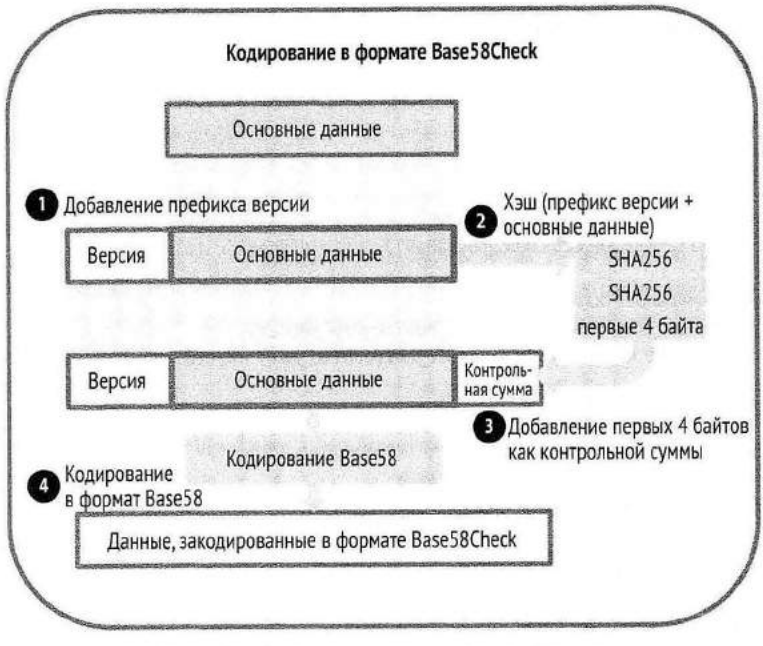


Рис. 4.6 ❖ Кодирование в Base58Check, формат Base58 с добавлением версии и контрольной суммы для корректного кодирования данных биткойна, исключающего неоднозначность

Таблица 4.1. Кодировка Base58Check: примеры префиксов версий и соответствующих итоговых символов

Тип	Префикс версии (в шестнадцатеричном формате)	Итоговый префикс Base58
Биткойн-адрес	0x00	1
Адрес платежного скрипта Pay-to-Script-Hash	0x05	3
Адрес для тестирования Bitcoin Testnet	0x6F	m или n
Секретный ключ WIF	0x80	5, K или L
Секретный ключ, зашифрованный по BIP-38	0x0142	6P
Расширенный открытый ключ по BIP-32	0x0488B21E	xpub

Рассмотрим полный процесс создания биткойн-адреса: от секретного ключа к открытому ключу (точка на эллиптической кривой), затем к дважды хэшированному адресу и, наконец, к процедуре кодирования в формате Base58Check.

Код на языке C++ в примере 4.3 подробно показывает этот процесс по шагам, от секретного ключа до биткойн-адреса, закодированного в формате Base58Check. В коде примера используются некоторые вспомогательные функции из библиотеки libbitcoin, представленной в разделе «Прочие клиенты, библиотеки и инструментальные пакеты» главы 3.

Пример 4.3 ❖ Создание биткойн-адреса, закодированного в формате Base58Check, из секретного ключа

```
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Личный секретный ключ.
    bc::ec_secret secret;
    bool success = bc::decode_base16(secret,
        "038109007313a5807b2eccc082c8c3fbb988a973cacf1a7df9ce725c31b14776");
    assert(success);
    // Вычисление открытого ключа.
    bc::ec_point public_key = bc::secret_to_public_key(secret);
    std::cout << "Public key: " << bc::encode_hex(public_key) << std::endl;

    // Создание биткойн-адреса
    // Обычно можно использовать:
    //   bc::payment_address payaddr;
    //   bc::set_public_key(payaddr, public_key);
    //   const std::string address = payaddr.encoded();

    // Вычисление хэш-значения открытого ключа для создания адреса P2PKH
    const bc::short_hash hash = bc::bitcoin_short_hash(public_key);

    bc::data_chunk unencoded_address;
    // Резервируется 25 байтов
    //   [ version:1  ]
    //   [ hash:20    ]
    //   [ checksum:4 ]
    unencoded_address.reserve(25);
    // Байт версии 0 - обычный BTC-адрес (P2PKH)
    unencoded_address.push_back(0);
    // Хэширование данных
    bc::extend_data(unencoded_address, hash);
    // Контрольная сумма вычисляется по хэшированным данным, потом добавляются 4 байта из хэша
    bc::append_checksum(unencoded_address);
    // Результат должен быть закодирован в формате Bitcoin Base58
    assert(unencoded_address.size() == 25);
    const std::string address = bc::encode_base58(unencoded_address);

    std::cout << "Address: " << address << std::endl;
    return 0;
}
```

В этом коде используется предварительно определенный секретный ключ для вычисления одного и того же биткойн-адреса при каждом запуске программы, как показано в примере 4.4.

Пример 4.4 ❖ Компиляция и запуск кода addr.cpp

```
# Компиляция кода addr.cpp
$ g++ -o addr addr.cpp $(pkg-config --cflags --libs libbitcoin)
# Запуск программы addr
$ ./addr
Public key: 0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK
```

Форматы ключей

И секретные, и открытые ключи могут быть представлены в нескольких различных форматах. Все эти представления кодируют одно и то же число, даже если выглядят по-разному. Эти форматы в основном используются для повышения удобочитаемости людьми и записи ключей без ошибок.

Форматы секретных ключей

Секретный ключ может быть представлен в нескольких различных форматах, соответствующих одному и тому же 256-битовому числу. В табл. 4.2 приведены три наиболее часто используемых формата для представления секретных ключей. Различные форматы применяются в разных условиях. Шестнадцатеричный и исходный бинарный форматы используются в программах и редко предъявляются пользователям. Формат WIF предназначен для экспорта/импорта ключей между кошельками и часто встречается в представлениях QR-кода (штрихового кода) секретных ключей.

Таблица 4.2. Представления секретных ключей (форматы кодирования)

Тип	Префикс	Описание
Исходный (raw)	Нет	32 байта
Шестнадцатеричный (hex)	Нет	64 шестнадцатеричные цифры
WIF	5	Кодировка Base58Check: Base58 с префиксом версии от 128-битовой и 32-битовой контрольной суммы
WIF сжатый	K или L	Аналогично WIF, но с добавлением суффикса 0x01 перед кодированием

В табл. 4.3 показан секретный ключ, закодированный в этих трех форматах.

Таблица 4.3. Пример: один и тот же ключ в различных форматах

Формат	Секретный ключ
Шестнадцатеричный (hex)	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8fc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF сжатый	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Все показанные выше представления являются различными способами записи одного и того же числа – секретного ключа. Они выглядят по-разному, но каждый формат может быть легко преобразован в другой. Следует отметить, что исходный бинарный (raw binary) формат не показан в табл. 4.3, так как все

кодировки, показанные выше, по умолчанию не являются исходными бинарными данными.

Воспользуемся командой `wif-to-ec` из проводника Bitcoin Explorer (см. приложение Ж), чтобы проверить, действительно ли оба формата WIF представляют один и тот же секретный ключ:

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

Декодирование из формата Base58Check

Команды проводника Bitcoin Explorer (см. приложение Ж) упрощают написание скриптов командной оболочки и конвейеров в командной строке для обработки ключей, адресов и транзакций биткойна. Можно воспользоваться проводником Bitcoin Explorer для декодирования из формата Base58Check в командной строке.

Применим команду `base58check-decode` для декодирования несжатого ключа:

```
$ bx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
wrapper
{
    checksum 4286807748
    payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
    version 128
}
```

Результат содержит ключ как значение параметра payload, префикс версии WIF 128 и контрольную сумму.

Отметим, что к значению payload для сжатого ключа добавлен суффикс `01`, означающий, что генерируемый открытый ключ должен быть сжатым:

```
$ bx base58check-decode KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
wrapper
{
    checksum 2339607926
    payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
    version 128
}
```

Кодирование из шестнадцатеричного формата в Base58Check

Для кодирования в формат Base58Check (операция, обратная приведенной выше команде) воспользуемся командой `base58check-encode` из проводника Bitcoin Explorer (см. приложение Ж) и возьмем в качестве параметра шестнадцатеричный секретный ключ, за которым следует префикс версии WIF 128:

```
bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd --version 128
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Кодирование из шестнадцатеричного формата (сжатого ключа) в Base58Check

Для кодирования в формат Base58Check «сжатого» секретного ключа (см. раздел «Сжатые секретные ключи» ниже в этой главе) добавляем суффикс 01 к шестнадцатеричному ключу, затем кодируем ключ, как в предыдущем разделе:

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version 128
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Полученный результат в WIF-сжатом формате начинается с символа K. Это означает, что секретный ключ внутри закодированных данных имеет суффикс 01 и будет использован только для генерации сжатых открытых ключей (см. раздел «Сжатые открытые ключи» ниже в этой главе).

Форматы открытых ключей

Открытые ключи также представляются различными способами, обычно как сжатые (compressed) или несжатые (uncompressed) открытые ключи.

Как мы видели ранее, открытый ключ – это точка на эллиптической кривой с координатами (x, y). Обычно она представлена с префиксом 04, за которым следуют два 256-битовых числа: координаты *x* и *y* соответственно. Префикс 04 используется, чтобы отличить несжатые открытые ключи от сжатых открытых ключей, которые начинаются с префикса 02 или 03.

Открытый ключ, сгенерированный по секретному ключу, созданному ранее, показан в виде пары координат:

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

Ниже приведен этот же открытый ключ в виде 520-битового числа (130 шестнадцатеричных цифр) с префиксом 04, за которым следуют координаты, то есть 04 x y:

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

Сжатые открытые ключи

Сжатые открытые ключи появились в биткойне для уменьшения размеров транзакций и экономии дискового пространства на узлах, которые хранят базу данных структуры блокчейна биткойн-системы. В большинство транзакций включается открытый ключ, необходимый для проверки подлинности личности владельца и права на расходование биткойнов. Каждый открытый ключ имеет длину 520 битов (префикс + *x* + *y*), что при умножении на несколько сотен транзакций в блоке или на десятки тысяч транзакций в день добавляет значительный объем данных в структуру блокчейна.

Как мы уже видели в разделе «Открытые ключи» этой главы, открытый ключ представляет собой точку (*x*,*y*) на эллиптической кривой. Поскольку

кривая выражается математической функцией, точка на этой кривой представляет решение соответствующего уравнения, следовательно, если нам известна координата *x*, то можно вычислить координату *y*, решив уравнение $y^2 \bmod p = (x^3 + 7) \bmod p$. Это позволяет хранить только координату *x* точки ключа, исключая координату *y*, тем самым уменьшая размер ключа и пространство, требуемое для его хранения на 256 битов. Почти 50-процентное сокращение размера в каждой транзакции позволяет хранить гораздо больший объем данных.

Несжатые открытые ключи имеют префикс `04`, в отличие от них, сжатые открытые ключи начинаются с префикса `02` или `03`. Выясним, почему возможны два различных префикса: так как в левой части уравнения находится выражение y^2, решением для *y* является квадратный корень, дающий положительное и отрицательное значения. В визуальном представлении это означает, что вычисляемая координата *y* может располагаться выше или ниже оси *X*. Таким образом, несмотря на то что координата *y* пропускается, необходимо хранить ее знак (плюс или минус), другими словами, необходимо запомнить, располагается ли точка выше или ниже оси *X*, потому что каждый вариант представляет совершенно различные точки, соответственно, и различные открытые ключи. При вычислениях на эллиптической кривой в терминах бинарной арифметики над конечным полем простого порядка *p* координата *y* является четной или нечетной, что соответствует знаку плюс/минус, как указано выше. Таким образом, чтобы различать два возможных значения *y*, сжатый открытый ключ сохраняется с префиксом `02`, если значение *y* четно, или с префиксом `03`, если оно нечетно, позволяя программному обеспечению правильно выводить значение координаты *y* по значению координаты *x* и «распаковывать» открытый ключ в виде полных координат точки. Процесс сжатия открытого ключа показан на рис. 4.7.

Здесь показан тот же самый открытый ключ, сгенерированный ранее, в виде сжатого открытого ключа, сохраненного в 264 битах (66 шестнадцатеричных цифр) с префиксом 03, соответствующим нечетной координате *y*:

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Этот сжатый открытый ключ соответствует тому же ранее созданному секретному ключу, то есть он сгенерирован на основе того же секретного ключа. Тем не менее он отличается по внешнему виду от несжатого открытого ключа. Но более важно то, что если выполнить преобразование этого сжатого открытого ключа в биткойн-адрес с помощью функции двойного хэширования (`RIPEMD160(SHA256(K))`), то в результате получится другой биткойн-адрес. Это может показаться неожиданным, так как означает, что один секретный ключ позволяет вычислить открытый ключ, представленный в двух различных форматах (сжатый и несжатый), которые дают два разных биткойн-адреса. И все же этот секретный ключ одинаков для обоих биткойн-адресов.

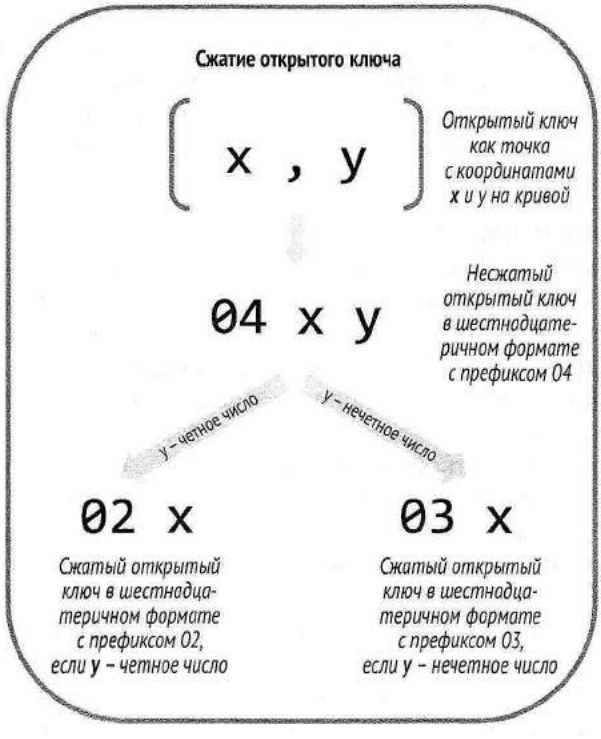


Рис. 4.7 ❖ Схема сжатия открытого ключа

Сжатые открытые ключи постепенно становятся форматом по умолчанию для всех пользователей биткойна, и это оказывает значительное воздействие на сокращение размеров транзакций, следовательно, и на уменьшение размера структуры данных блокчейна. Но пока еще не все клиенты поддерживают формат сжатых открытых ключей. Более новые клиенты, поддерживающие формат сжатых открытых ключей, вынуждены работать с транзакциями от более старых клиентов, которые не поддерживают этот формат. Это особенно важно, если приложение кошелька импортирует секретные ключи из другого приложения биткойн-кошелька, потому что новый кошелек должен просканировать структуру данных блокчейна в поисках транзакций, соответствующих этим импортируемым ключам. А какие биткойн-адреса должно искать приложение нового кошелька? Биткойн-адреса, полученные из несжатых открытых ключей или из сжатых открытых ключей? И те, и другие являются корректными биткойн-адресами и могут быть подписаны одним секретным ключом, но все же это различные адреса.

Для решения описанной проблемы при экспортировании секретных ключей из какого-либо кошелька формат WIF (предназначенный для представления секретных ключей) в новых биткойн-кошельках реализован по-другому, указывая, что эти секретные ключи использовались для генерации сжатых от-

крытых ключей, следовательно, и для сжатых биткойн-адресов. Это позволяет кошельку, выполняющему импортирование, различать секретные ключи из старых и новых кошельков и проводить поиск в структуре данных блокчейна транзакций с биткойн-адресами, соответствующими несжатым или сжатым открытым ключам соответственно. В следующем разделе мы более подробно рассмотрим, как это работает.

Сжатые секретные ключи

Как ни странно, термин «сжатый секретный ключ» искажает истину, так как экспортируемый в сжатом формате WIF секретный ключ в действительности на один байт длиннее, чем «несжатый» секретный ключ. Причина в том, что сжатый секретный ключ содержит дополнительный однобайтный суффикс (показанный как шестнадцатеричное значение `01` в табл. 4.4), удостоверяющий, что этот ключ взят из современной версии кошелька и должен использоваться для генерации только сжатых открытых ключей. Сами по себе секретные ключи не сжимаются и вообще не могут быть сжаты. Термин «сжатый секретный ключ» в действительности означает «секретный ключ, по которому должны генерироваться только сжатые открытые ключи». Тогда термин «несжатый секретный ключ» в действительности означает «секретный ключ, по которому должны генерироваться только несжатые открытые ключи». Чтобы в дальнейшем избежать путаницы, следует четко обозначать формат экспортирования как «WIF-compressed» или «WIF», а не называть секретный ключ просто «сжатым».

В табл. 4.4 показан один и тот же секретный ключ, закодированный в форматах WIF и WIF-compressed (сжатом).

Таблица 4.4. Пример: один секретный ключ в разных форматах

Формат	Секретный ключ
Шестнадцатеричный (Hex)	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Шестнадцатеричный сжатый (Hex-compressed)	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
WIF сжатый (WIF-compressed)	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Отметим, что секретный ключ в сжатом шестнадцатеричном формате в конце содержит один дополнительный байт (шестнадцатеричное значение 01). Так как префикс версии в кодировке Base58 одинаков (0x80) для форматов WIF и WIF-compressed, добавление одного байта в конец числа приводит к изменению первого символа в кодировке Base58 с 5 на K или L. Для кодировки Base58 это можно считать аналогом различий между числами 100 и 99 в десятичной системе счисления. Число 100 на одну цифру длиннее числа 99, но оно имеет префикс 1 вместо префикса 9. То есть изменение длины непосредственно влияет на префикс. В кодировке Base58 префикс 5 изменяется на K или L, поскольку длина числа увеличилась на один байт.

Следует помнить, что эти форматы не являются взаимозаменяемыми. В более новых версиях кошельков с поддержкой сжатых открытых ключей секретные ключи должны экспортироваться только в формате WIF-compressed (с префиксом K или L). Если вы работаете со старой реализацией кошелька, не использующей сжатые открытые ключи, то секретные ключи должны экспортироваться только в формате WIF (с префиксом 5). Здесь задача состоит в том, чтобы сообщить кошельку, выполняющему импортирование секретных ключей, должен ли он искать в структуре блокчейна сжатые или несжатые открытые ключи и адреса.

Если в биткойн-кошельке есть возможность применения сжатых открытых ключей, он будет использовать их во всех транзакциях. Секретные ключи в таком кошельке будут применяться для генерации открытых ключей как точек на эллиптической кривой, координаты которых будут сжаты. С помощью сжатых открытых ключей будут формироваться биткойн-адреса, включаемые в транзакции. При экспорте из кошелька новой версии с реализацией сжатых открытых ключей формат WIF изменяется посредством добавления однобайтного суффикса `01` к секретному ключу. Закодированный в формате Base58Check секретный ключ обозначается как «WIF-compressed» и начинается с букв K или L вместо цифры 5, обозначающей обычный (несжатый) формат WIF ключей из более старых версий кошельков.

«Сжатые секретные ключи» – термин, искажающий действительность. Секретные ключи не сжимаются, а формат WIF-compressed означает, что такие ключи должны использоваться для генерации только сжатых открытых ключей и соответствующих им биткойн-адресов. Как ни странно, «сжатый» формат кодирования секретных ключей WIF-compressed на один байт длиннее из-за добавления суффикса 01, чтобы отличать его от «несжатого» формата.

Реализация ключей и адресов на языке Python

Самой полной и тщательно проработанной библиотекой поддержки биткойна на языке программирования Python является pybitcointools (https://github.com/vbuterin/pybitcointools), написанная Виталиком Бутериным (Vitalik Buterin). В примере 4.5 эта библиотека (импортируемая как bitcoin) используется для генерации и вывода ключей и адресов в различных форматах.

Пример 4.5 ❖ Генерация и представление в различных форматах ключей и адресов с использованием библиотеки pybitcointools

```
import bitcoin

# Генерация случайного секретного ключа
valid_private_key = False
while not valid_private_key:
    private_key = bitcoin.random_key()
    decoded_private_key = bitcoin.decode_privkey(private_key, 'hex')
    valid_private_key = 0 < decoded_private_key < bitcoin.N
```

```
print "Private Key (hex) is: ", private_key
print "Private Key (decimal) is: ", decoded_private_key

# Преобразование секретного ключа в формат WIF
wif_encoded_private_key = bitcoin.encode_privkey(decoded_private_key, 'wif')
print "Private Key (WIF) is: ", wif_encoded_private_key

# Добавление суффикса "01" для обозначения сжатого секретного ключа
compressed_private_key = private_key + '01'
print "Private Key Compressed (hex) is: ", compressed_private_key

# Генерация формата WIF из сжатого секретного ключа (WIF-compressed)
wif_compressed_private_key = bitcoin.encode_privkey(
    bitcoin.decode_privkey(compressed_private_key, 'hex'), 'wif')
print "Private Key (WIF-Compressed) is: ", wif_compressed_private_key

# Умножение базовой точки генерации G на эллиптической кривой на секретный ключ
# для получения точки открытого ключа
public_key = bitcoin.fast_multiply(bitcoin.G, decoded_private_key)
print "Public Key (x,y) coordinates is:", public_key

# Кодирование в шестнадцатеричном формате с префиксом 04
hex_encoded_public_key = bitcoin.encode_pubkey(public_key,'hex')
print "Public Key (hex) is:", hex_encoded_public_key

# Сжатие открытого ключа, выбор префикса в зависимости от четности или нечетности координаты y
(public_key_x, public_key_y) = public_key
if (public_key_y % 2) == 0:
    compressed_prefix = '02'
else:
    compressed_prefix = '03'
hex_compressed_public_key = compressed_prefix + bitcoin.encode(public_key_x, 16)
print "Compressed Public Key (hex) is:", hex_compressed_public_key

# Генерация биткойн-адреса из открытого ключа
print "Bitcoin Address (b58check) is:", bitcoin.pubkey_to_address(public_key)

# Генерация сжатого биткойн-адреса из сжатого открытого ключа
print "Compressed Bitcoin Address (b58check) is:", \
    bitcoin.pubkey_to_address(hex_compressed_public_key)
```

В примере 4.6 показан вывод результата после выполнения этого кода.

Пример 4.6 ❖ Выполнение программы key-to-address-ecc-example.py

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
 3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (decimal) is:
 26563230048437957592232553826663696440606756685920117476832299673293013768870
Private Key (WIF) is:
 5JG9hT3beGTJuUAmCQEmNaxAuMacCTfXuw1R3FCXig23RQHMr4K
Private Key Compressed (hex) is:
 3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa601
Private Key (WIF-Compressed) is:
 KyBsPXxTuVD82av65KZkrGrWi5qLMah5SdNq6uftawDbgKa2wv6S
```

```
Public Key (x,y) coordinates is:
 (41637322786646325214887832269588396900663353932545912953362782457239403430124L,
 16388935128781238405526710466724741593761085120864331449066658622400339362166L)
Public Key (hex) is:
 045c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec
243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
 025c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00ec
Bitcoin Address (b58check) is:
 1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
 14cxpo3MBCYYWCgF74SWTdcmxipnGUsPw3
```

В примере 4.7 приведен другой код с использованием библиотеки Python ECDSA, обеспечивающей поддержку математики эллиптических кривых. Здесь не применяются какие-либо библиотеки, специализированные для биткойна.

Пример 4.7 ❖ Скрипт, демонстрирующий применение математики эллиптических кривых для вычисления ключей биткойна

```
import ecdsa
import os
from ecdsa.util import string_to_number, number_to_string

# secp256k1, http://www.oid-info.com/get/1.3.132.0.10
_p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2FL
_r = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141L
_b = 0x0000000000000000000000000000000000000000000000000000000000000007L
_a = 0x0000000000000000000000000000000000000000000000000000000000000000L
_Gx = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798L
_Gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8L
curve_secp256k1 = ecdsa.ellipticcurve.CurveFp(_p, _a, _b)
generator_secp256k1 = ecdsa.ellipticcurve.Point(curve_secp256k1, _Gx, _Gy, _r)
oid_secp256k1 = (1, 3, 132, 0, 10)
SECP256k1 = ecdsa.curves.Curve("SECP256k1", curve_secp256k1, generator_secp256k1,
oid_secp256k1)
ec_order = _r

curve = curve_secp256k1
generator = generator_secp256k1

def random_secret():
    convert_to_int = lambda array: int("".join(array).encode("hex"), 16)

    # Сбор 256 битов случайных данных из криптографически надежного генератора случайных чисел
    # операционной системы
    byte_array = os.urandom(32)

    return convert_to_int(byte_array)

def get_point_pubkey(point):
    if point.y() & 1:
        key = '03' + '%064x' % point.x()
    else:
        key = '02' + '%064x' % point.x()
```

```
    return key.decode('hex')

def get_point_pubkey_uncompressed(point):
    key = '04' + \
          '%064x' % point.x() + \
          '%064x' % point.y()
    return key.decode('hex')

# Генерация нового секретного ключа
secret = random_secret()
print "Secret: ", secret

# Вычисление точки открытого ключа
point = secret * generator
print "EC point:", point

print "BTC public key:", get_point_pubkey(point).encode("hex")

# По точке с координатами (x, y) можно создать требуемый объект:
point1 = ecdsa.ellipticcurve.Point(curve, point.x(), point.y(), ec_order)
assert point1 == point
```

В примере 4.8 показан вывод результата выполнения этого скрипта.

В примере 4.7 используется функция os.urandom, использующая криптографически надежный генератор случайных чисел (CSRNG), предоставляемый операционной системой. В Unix-подобных операционных системах, таких как Linux, это соответствует псевдоустройству /dev/urandom. В ОС Windows генератор вызывает CryptGenRandom(). Если подходящий источник случайных чисел не найден, то возникает ошибка NotImplementedError. Генератор случайных чисел в приведенном выше примере используется исключительно в демонстрационных целях, но его не следует применять для генерации реально эксплуатируемых ключей биткойна, так как эта реализация не обеспечивает приемлемого уровня безопасности.

Пример 4.8 ❖ Установка библиотеки Python ECDSA и запуск скрипта ec_math.py

```
$ # Установка менеджера пакетов Python PIP
$ sudo apt-get install python-pip
$ # Установка библиотеки Python ECDSA
$ sudo pip install ecdsa
$ # Запуск скрипта
$ python ec-math.py
Secret:
38090835015954358862481132628887443905906204995912378278060168703580660294000
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

Усовершенствованные ключи и адреса

В следующих разделах мы рассмотрим усовершенствованные формы ключей и адресов, такие как зашифрованные секретные ключи, адреса скриптов и адреса мультиподписей, «престижные» (vanity) кошельки и бумажные кошельки.

Зашифрованные секретные ключи (BIP-38)

Секретные ключи непременно должны храниться в тайне. Требование к конфиденциальности (confidentiality) секретных ключей общеизвестно, но соблюдения его достаточно трудно добиться на практике, поскольку это требование вступает в конфликт с не менее важным требованием: доступностью (availability). Сохранить секретный ключ в тайне намного труднее, если необходимо организовать хранение резервных копий секретного ключа, чтобы избежать его потери. Секретный ключ, хранимый в кошельке и защищенный паролем, возможно, защищен вполне надежно, но для самого кошелька необходимы резервные копии. Иногда нужно переместить ключи из одного кошелька в другой, например при обновлении или замене программного обеспечения. Кроме того, резервные копии секретных ключей могут храниться в виде распечаток на бумаге (см. раздел «Бумажные кошельки» ниже в этой главе) или на внешних устройствах, таких как USB флеш-накопитель. Ну а если сама резервная копия похищена или потеряна? Эти конфликтующие требования к обеспечению безопасности привели к созданию соответствующего универсального стандарта шифрования секретных ключей переносимым методом, который могли бы распознавать многие различные кошельки и клиенты биткойна, стандартизированные в соответствии с предложением BIP-38 (см. приложение B).

Документ BIP-38 предлагает общий стандарт шифрования секретных ключей с помощью парольной фразы (passphrase) и кодирования с помощью Base58Check для обеспечения безопасного их хранения на устройствах резервного копирования, для защищенной передачи между кошельками, а также для хранения в любых других условиях, при которых возможно раскрытие ключа. Этот стандарт шифрования использует стандарт AES (Advanced Encryption Standard), установленный Национальным институтом стандартов США (NIST) и широко применяемый для шифрования данных для коммерческих и военных приложений.

Схема зашифрования по BIP-38 определяет в качестве входных данных секретный ключ биткойна, обычно закодированный в формате WIF как строка в кодировке Base58Check с префиксом 5. В дополнение к этому схема зашифрования по BIP-38 принимает парольную фразу – длинный пароль, – обычно состоящий из нескольких слов или представляющий собой сложную строку из алфавитно-цифровых символов. Результатом выполнения схемы зашифрования по BIP-38 является зашифрованный секретный ключ в кодировке Base58Check, начинающийся с префикса `6P`. Если вы видите ключ, начинающийся с `6P`, то он зашифрован и требует парольную фразу для обратного преобразования (расшифрования) в секретный ключ в формате WIF (с префиксом 5), который можно использовать в любом кошельке. Сейчас многие приложения кошельков распознают зашифрованные по схеме BIP-38 секретные ключи и предлагают пользователю ввести парольную фразу для расшифрования и импортирования такого ключа. Кроме того, для расшифрования BIP-38 ключей можно воспользоваться приложениями независимых производителей, на-

пример чрезвычайно полезным приложением на основе браузера Bit Address (http://bitaddress.org) (вкладка Wallet Details).

Наиболее часто шифрование секретных ключей по схеме BIP-38 используется для бумажных кошельков, служащих для сохранения резервных копий секретных ключей на обычном листе бумаги. Если пользователь выбирает достаточно сложную парольную фразу, то бумажный кошелек с секретными ключами, зашифрованными по схеме BIP-38, защищен вполне надежно и является отличным способом хранения биткойнов в режиме офлайн (этот способ также называют «холодным хранением» (cold storage)).

Проверьте зашифрованные ключи в табл. 4.5 на сайте http://bitaddress.org, чтобы убедиться в возможности получения расшифрованного ключа после ввода парольной фразы.

Таблица 4.5. Пример секретного ключа, зашифрованного по схеме BIP-38

Секретный ключ (в формате WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Парольная фраза	MyTestPassphrase
Зашифрованный ключ (по схеме BIP-38)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCdctLJ3z5yxE87MobKoXdTsJ

Адреса скриптов Pay-to-Script Hash (P2SH) и адреса мультиподписей

Как нам уже известно, обычный биткойн-адрес начинается с цифры 1 и создан на основе открытого ключа, который, в свою очередь, сгенерирован из секретного ключа. Несмотря на то что каждый может переслать биткойн на любой адрес, начинающийся с 1, этот биткойн может быть израсходован только после предоставления подписи секретного ключа и хэш-значения открытого ключа, соответствующих друг другу.

Биткойн-адреса, начинающиеся с цифры 3, – это адреса скриптов Pay-to-script hash (P2SH), иногда ошибочно называемые адресами мультиподписей (multisignature или multisig). Они определяют получателя транзакции биткойнов как хэш-значение скрипта вместо владельца открытого ключа. Эта функциональная возможность была введена в январе 2012 года по предложению BIP-16 (см. приложение B) и сразу же была принята и широко распространена, поскольку предоставляет дополнительную функциональность непосредственно для адреса. В отличие от транзакций, которые «передают» денежные средства на обычные биткойн-адреса, начинающиеся с 1 (их также обозначают как pay-to-public-key-hash (P2PKH)), денежные средства, пересылаемые на адреса, начинающиеся с 3, требуют несколько большего, нежели предоставления хэш-значения одного открытого ключа и подписи одного секретного ключа как доказательства права владения. Требования определяются во время создания адреса в специальном скрипте, поэтому все поступления на такой адрес будут обязаны выполнять их.

Адрес P2SH создается из скрипта транзакции, в котором определяется, кто может расходовать выходные данные транзакции (более подробно см. раздел

«Pay-to-Script-Hash (P2SH)» в главе 7). При кодировании адреса P2SH предполагается использование той же функции двойного хэширования, что и для создания обычного биткойн-адреса, только вместо открытого ключа она применяется к скрипту:

```
script hash = RIPEMD160(SHA256(script))
```

Полученный «хэш скрипта» представлен в кодировке Base58Check с префиксом версии 5, что в результате дает закодированный адрес, начинающийся с цифры 3. Пример адреса P2SH – `3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM` – может быть сгенерирован с помощью команд проводника Bitcoin Explorer `script-encode`, `sha256`, `ripemd160` и `base58check-encode` (см. приложение Ж) следующим образом:

```
$ echo dup hash160 [ 89abcdefabbaabbaabbaabbaabbaabbaabbaabba ] equalverify
checksig > script
$ bx script-encode < script | bx sha256 | bx ripemd160 | bx base58check-encode
--version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

P2SH далеко не всегда является стандартной транзакцией с мультиподписью. Адрес P2SH весьма часто представляет скрипт мультиподписи, но, кроме того, он может представлять скрипт, содержащий код других типов транзакций.

Адреса мультиподписей и P2SH

В настоящее время наиболее частой реализацией функции P2SH является скрипт адреса мультиподписи. Из названия понятно, что такой скрипт требует более одной подписи для подтверждения права владения, следовательно, и расходования денежных средств. Функциональная возможность формирования мультиподписей в биткойн-системе предназначена для назначения обязательного требования M подписей (часто обозначаемого термином «пороговый уровень» (threshold)) от N ключей в сумме, известного как требование мультиподписи M-of-N, где M меньше или равно N. Например, владелец кафе Боб из главы 1 мог бы использовать адрес мультиподписи, требующий 1 из 2 подписей от ключа, принадлежащего ему, и от ключа, принадлежащего его супруге, тем самым подтверждая, что любая из этих возможных подписей дает право расходования выходных данных транзакций, связанных с таким адресом. Это отчасти похоже на «общий счет» (joint account), применяемый в банковской сфере, когда любой из супругов имеет право расходовать денежные средства, заверяя операции единственной подписью. Гопеш, веб-дизайнер, которому Боб оплатил создание веб-сайта, может иметь для своего бизнеса адрес мультиподписи 2-of-3, который гарантирует, что расходование денежных средств невозможно до тех пор, пока по меньшей мере два бизнес-партнера не подпишут транзакцию.

Более подробно создание транзакций, расходующих средства с адресов P2SH (и с адресов мультиподписей), будет рассматриваться в главе 6.

«Престижные» адреса

«Престижные» (vanity) адреса – это корректные биткойн-адреса, которые содержат осмысленные сообщения. Например, `1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33` – это допустимый корректный адрес, содержащий буквы, образующие слово Love, как первые четыре символа в кодировке Base58. Создание «престижных» адресов требует генерации и проверки миллиардов потенциальных секретных ключей до тех пор, пока не будет найдена желаемая комбинация. Несмотря на то что алгоритм генерации «престижных» адресов до некоторой степени оптимизирован, в сущности процесс включает выбор случайного секретного ключа, генерацию открытого ключа, генерацию биткойн-адреса и визуальную проверку последнего. Эта процедура повторяется миллиарды раз до тех пор, пока не будет найден подходящий адрес.

После того как требуемый «престижный» адрес найден, секретный ключ, из которого был сгенерирован этот адрес, может использоваться владельцем для расходования биткойнов точно так же, как и любой другой адрес. «Престижные» адреса не более и не менее безопасны, чем обычный биткойн-адрес. Как и все биткойн-адреса, они зависят от методики эллиптической криптографии (Elliptic Curve Cryptography, ECC) и от алгоритма хэширования SHA. Найти секретный ключ для адреса, начинающегося с «престижной» комбинации букв, не проще, чем любой другой конкретный адрес.

В главе 1 мы познакомились с Эухенией (Eugenia), директором фонда помощи детям на Филиппинах. Предположим, что Эухения занимается организацией сбора средств в биткойнах и хочет использовать «престижный» биткойн-адрес, обозначающий главную цель фонда. Эухения должна создать «престижный» адрес, начинающийся с символов «1Kids», чтобы объявить всем, для чего организован сбор средств. Рассмотрим подробнее, как можно сформировать такой адрес и что это означает для безопасности фонда Эухении.

Генерация «престижных» адресов

Важно понимать, что биткойн-адрес – это просто число, представленное символами алфавита Base58. Поиск образца, подобного «1Kids», может рассматриваться как поиск адреса в диапазоне от `1Kids11111111111111111111111111111` до `1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzz`. В этом диапазоне расположено приблизительно 58^{29} (около 1.4×10^{51}) адресов, начинающихся на «1Kids». В табл. 4.6 показан диапазон адресов с префиксом 1Kids.

Таблица 4.6. Диапазон «престижных» адресов, начинающихся с «1Kids»

От	1Kids11111111111111111111111111111
	1Kids11111111111111111111111111112
	1Kids11111111111111111111111111113
	...
До	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzz

Будем рассматривать образец «1Kids» как число и попробуем определить, как часто можно обнаружить этот образец в биткойн-адресах (см. табл. 4.7). Обычный настольный персональный компьютер без специализированной аппаратуры может анализировать около 100 000 ключей в секунду.

Таблица 4.7. Частота появления «престижного» образца (1KidsCharity) и среднее время его поиска на обычном настольном персональном компьютере

Длина	Образец	Частота появления	Среднее время поиска
1	1K	1 из 58 ключей	менее 1 миллисекунды
2	1Ki	1 из 3364	50 миллисекунд
3	1Kid	1 из 195 000	менее 2 секунд
4	1Kids	1 из 11 миллионов	1 минута
5	1KidsC	1 из 656 миллионов	1 час
6	1KidsCh	1 из 38 миллиардов	2 дня
7	1KidsCha	1 из 2.2 триллиона	3–4 месяца
8	1KidsChar	1 из 128 триллионов	13–18 лет
9	1KidsChari	1 из 7 квадриллионов	800 лет
10	1KidsCharit	1 из 400 квадриллионов	46 000 лет
11	1KidsCharity	1 из 23 квинтиллионов	2.5 миллиона лет

По этой таблице можно понять, что Эухения не сможет создать «престижный» адрес 1KidsCharity за приемлемое время, даже если она получит доступ к нескольким тысячам компьютеров. Каждый дополнительный символ увеличивает сложность задачи с коэффициентом 58. Для поиска образцов, содержащих более семи символов, обычно требуется специализированная аппаратура, например настольные компьютеры в особой комплектации с несколькими графическими процессорами (GPU) (то есть с несколькими мощными видеокартами). Такие компьютеры часто представляют собой так называемые «агрегаты» или «фермы», которые больше не пригодны для майнинга биткойнов, но могут использоваться для поиска «престижных» адресов. Поиск на системах с несколькими GPU выполняется на несколько порядков быстрее, чем на настольных компьютерах с обычными процессорами.

Другой способ получения «престижного» адреса – передача вычислительной работы в пул майнеров «престижа», например в Vanity Pool (http://vanitypool.appspot.com). Подобный пул представляет собой сервис, позволяющий обладателям мощных графических процессоров зарабатывать биткойны, выполняя поиск «престижных» адресов для других пользователей. За небольшую плату (0.01 биткойна, или приблизительно 5 долларов на момент написания этой книги) Эухения может передать в такой пул задачу поиска адреса, начинающегося с семи заданных символов, и получить результат через несколько часов, не затрачивая месяцы на решение с помощью обычного настольного компьютера.

Генерацию «престижного» адреса можно рассматривать как учебную задачу поиска методом полного перебора (или «грубой силы»): выбор случайного ключа, проверка сгенерированного адреса на совпадение с заданным образ-

цом, повторение до получения желаемого результата. В примере 4.9 показан код «майнера престижного адреса», программы на языке C++, выполняющей поиск «престижных» адресов. В примере используется библиотека libbitcoin, описанная в разделе «Прочие клиенты, библиотеки и инструментальные пакеты» главы 3.

Пример 4.9 ❖ Майнер «престижных» адресов

```
#include <bitcoin/bitcoin.hpp>

// Искомая строка
const std::string search = "1kid";

// Генерация случайного секретного ключа. Случайное число размером в 32 байта
bc::ec_secret random_secret(std::default_random_engine& engine);
// Получение биткойн-адреса из секретного ключа
std::string bitcoin_address(const bc::ec_secret& secret);
// Сравнение со строкой поиска без учета регистра символов
bool match_found(const std::string& address);

int main()
{
    // random_device в ОС Linux использует "/dev/urandom"
    // ВНИМАНИЕ: в зависимости от реализации этот генератор случайных чисел может оказаться
    // недостаточно надежным
    // Не используйте ключи, сгенерированные в этом примере, для реальной практики
    std::random_device random;
    std::default_random_engine engine(random());

    // Внимание: бесконечный цикл
    while (true)
    {
        // Генерация случайного секретного ключа
        bc::ec_secret secret = random_secret(engine);
        // Получение адреса
        std::string address = bitcoin_address(secret);
        // Проверка на совпадение со строкой поиска (1kid)
        if (match_found(address))
        {
            // Совпадение
            std::cout << "Found vanity address! " << address << std::endl;
            std::cout << "Secret: " << bc::encode_hex(secret) << std::endl;
            return 0;
        }
    }
    // Эта часть кода никогда не должна выполняться
    return 0;
}

bc::ec_secret random_secret(std::default_random_engine& engine)
{
    // Создание нового секретного ключа
    bc::ec_secret secret;
```

```
    // Итерация по байтам для установки случайного значения
    for (uint8_t& byte: secret)
        byte = engine() % std::numeric_limits<uint8_t>::max();
    // Возврат результата
    return secret;
}

std::string bitcoin_address(const bc::ec_secret& secret)
{
    // Преобразование секретного ключа в открытый ключ
    bc::ec_point pubkey = bc::secret_to_public_key(secret);
    // Формирование адреса
    bc::payment_address payaddr;
    bc::set_public_key(payaddr, pubkey);
    // Возврат закодированной формы адреса
    return payaddr.encoded();
}

bool match_found(const std::string& address)
{
    auto addr_it = address.begin();
    // Цикл поиска строки-образца - сравнение ее с символами полученного адреса,
    // переведенными в нижний регистр
    for (auto it = search.begin(); it != search.end(); ++it, ++addr_it)
        if (*it != std::tolower(*addr_it))
            return false;
    // Достигнут конец строки поиска - найдено совпадение с символами адреса
    return true;
}
```

В примере 4.10 используется источник `std::random_device`. В зависимости от реализации он может соответствовать криптографически надёжному генератору случайных чисел CSRNG, предоставляемому операционной системой. В Unix-подобных операционных системах, таких как Linux, это соответствует псевдоустройству `/dev/urandom`. Генератор случайных чисел в приведенном выше примере используется исключительно в демонстрационных целях, но его не следует применять для генерации реально эксплуатируемых ключей биткойна, так как эта реализация не обеспеспечивает приемлемого уровня безопасности.

Код приведенного выше примера необходимо скомпилировать с помощью любого компилятора C++ и скомпоновать (link) с библиотекой `libbitcoin` (которая должна быть установлена в системе). Для выполнения программы запустите выполняемый файл `vanity-miner` без параметров (см. пример 4.10), который попытается найти «престижный» адрес, начинающийся с символов 1kid.

Пример 4.10 ❖ Компиляция и выполнение примера vanity-miner

```
$ # Компиляция и компоновка исходного кода с помощью компилятора g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Выполнение программы-примера
$ ./vanity-miner
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
```

```
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Повторный запуск с целью получения другого результата
$ ./vanity-miner
Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Использование команды "time" для определения времени, затраченного на поиск результата
$ time ./vanity-miner
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

real  0m8.868s
user  0m8.828s
sys  0m0.035s
```

Программа-пример за несколько секунд находит совпадение с трехсимвольным образцом `kid`, а если воспользоваться Unix-командой `time`, то можно более точно определить затраченное время. Измените образец `pattern` в исходном коде и понаблюдайте, сколько времени потребуется для поиска четырехсимвольных или пятисимвольных образцов.

Безопасность «престижного» адреса

«Престижные» адреса могут использоваться как для улучшения, так и для ухудшения обеспечения безопасности – они поистине являются обоюдоострым клинком. Используемый для усиления защиты характерный узнаваемый адрес злоумышленникам труднее заменить на свой адрес, чтобы обмануть ваших клиентов и заставить их платить им, а не вам. К сожалению, «престижные» адреса также дают возможность создавать адреса, похожие (частично совпадающие) на любой случайный адрес или даже на другой «престижный» адрес, что способствует обману клиентов.

Эухения могла бы опубликовать случайно сгенерированный адрес (например, `1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy`) для приема пожертвований. Или сгенерировать «престижный» адрес, начинающийся с символов 1Kids, чтобы сделать его более заметным.

В обоих случаях одна из опасностей использования одного фиксированного адреса (вместо отдельных динамических адресов для каждого пожертвователя) состоит в том, что злоумышленник получает возможность проникнуть на ваш веб-сайт и заменить опубликованный адрес на свой, перенаправляя пожертвования себе. Если вы опубликовали адреса для пожертвований в нескольких различных местах, то пользователи могут визуально проверить адрес перед платежом, чтобы убедиться в том, что это тот же адрес, который они видели на вашем сайте, в ваших сообщениях электронной почты или в ваших рекламных материалах. При использовании случайного адреса, такого как `1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy`, обычный пользователь, вероятнее всего, проверит несколько первых символов «1J7mdg» и решит, что адрес правильный. С помощью генератора «престижных» адресов злоумышленник с намерением подменить адрес на похожий может быстро сформировать адреса, в ко-

торых несколько первых символов совпадают с первыми символами вашего адреса, как показано в табл. 4.8.

Таблица 4.8. Генерация «престижных» адресов, частично совпадающих со случайным адресом

Исходный случайный адрес	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
«Престижный» адрес (совпадение 4 символов)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
«Престижный» адрес (совпадение 5 символов)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
«Престижный» адрес (совпадение 6 символов)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Итак, действительно ли «престижный» адрес укрепляет защиту? Если Эухения генерирует «престижный» адрес `1Kids33q44erFfpeXrmDSz7zEqG2FesZEN`, то, вероятнее всего, пользователи будут проверять «престижное» слово и несколько символов, следующих за ним, например часть адреса «1Kids33». Это усложняет для злоумышленника задачу генерации адреса, совпадающего по меньшей мере в шести первых символах (на два длиннее), таким образом, затраты на такое решение возрастают в 3364 раза (58 × 58), по сравнению с затратами Эухении на поиск адреса с четырьмя заданными символами. По сути, затраты Эухении (или плата в пул «престижных» адресов) вынуждают злоумышленника искать более длинный совпадающий образец. Если Эухения платит пулу за генерацию 8-символьного «престижного» адреса, то злоумышленнику придется подбирать 10 символов, что практически невозможно сделать на обычном персональном компьютере и чрезвычайно трудно даже с помощью специализированной фермы или пула майнинга «престижных» адресов. Возможное для Эухении становится недостижимым для злоумышленника, особенно если предполагаемая выгода от подмены адреса недостаточна, чтобы покрыть затраты на генерацию требуемого «престижного» адреса.

Бумажные кошельки

Бумажные кошельки – это секретные ключи биткойна, напечатанные на бумаге. Часто в бумажный кошелек также включается соответствующий биткойн-адрес для удобства, но это не обязательно, поскольку адрес всегда можно вычислить по секретному ключу. Бумажные кошельки представляют собой весьма эффективный способ создания резервных копий или хранения биткойнов в режиме офлайн, также известный как «холодное хранение» (cold storage). Как механизм резервного копирования бумажный кошелек может обеспечить защиту от потери ключа, если что-либо случится с компьютером, например поломка жесткого диска, кража или случайное удаление данных. Если ключи в бумажном кошельке сгенерированы в режиме офлайн и никогда не хранились на компьютерной системе, то механизм «холодного хранения» обеспечивает более надежную защиту от взломщиков, кейлоггеров и прочих опасностей для компьютеров, подключенных к сетям.

Бумажные кошельки имеют различные формы, размеры и внешний вид, но самый простой вариант – это ключ и адрес, напечатанные на бумаге. В табл. 4.9 показана простейшая форма бумажного кошелька.

Таблица 4.9. Простейшая форма бумажного кошелька – распечатка биткойн-адреса и секретного ключа

Общедоступный адрес	Секретный ключ (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

Бумажные кошельки можно легко создать с помощью инструментов, подобных генератору, работающему на стороне клиента, написанному на JavaScript и расположенному на сайте bitaddress.org. Эта страница содержит код, необходимый для генерации ключей и бумажных кошельков даже при полном отключении от Интернета. Чтобы воспользоваться этим инструментом, сохраните соответствующую HTML-страницу на локальном диске или на внешнем флеш-накопителе. Отключитесь от Интернета и откройте сохраненный файл в браузере. А лучше перезагрузите компьютер, используя «чистую» операционную систему, например версию ОС Linux, загружаемую и работающую с CD-ROM. Любой ключ, сгенерированный этим инструментом в режиме офлайн, можно распечатать на локальном принтере с подключением по USB-кабелю (но не с беспроводным подключением). Таким способом создаются бумажные кошельки, ключи которых существуют только на бумаге и никогда не хранились на какой-либо системе, подключенной к сети. Положите созданные бумажные кошельки в несгораемый шкаф (сейф) и «отправляйте» биткойны на адреса, указанные в этих кошельках. Это и будет реализация простого, но чрезвычайно эффективного «холодного хранения». На рис. 4.8 показан бумажный кошелек, сгенерированный с помощью страницы сайта bitaddress.org.

Рис. 4.8 ❖ Пример бумажного кошелька, сгенерированного с помощью страницы сайта bitaddress.org

Основной недостаток системы простых бумажных кошельков – распечатанные ключи могут быть похищены. Злоумышленник, получивший доступ к бумажному кошельку, может его украсть или сфотографировать ключи и перехватить управление биткойнами, защищаемыми этими ключами. В более хитроумной системе хранения бумажных кошельков используются секретные

ключи, зашифрованные в соответствии с документом BIP-38. Ключи, распечатанные в бумажном кошельке, защищены парольной фразой, которую владелец должен запомнить. Без парольной фразы зашифрованные ключи бесполезны. Вариант с бумажным кошельком, защищенным парольной фразой, более надежен, поскольку и в таком кошельке ключи никогда не хранились в режиме онлайн, и для применения их необходимо действительно извлечь из сейфа или другого физически защищенного хранилища. На рис. 4.9 показан бумажный кошелек с зашифрованным секретным ключом (BIP-38), созданный с помощью страницы сайта bitaddress.org.

Рис. 4.9 ❖ Пример бумажного кошелька, сгенерированного с помощью страницы сайта bitaddress.org. Парольная фраза: «test»

Вкладывать денежные средства в бумажный кошелек можно многократно, но снять их можно только один раз, потратив всю сумму полностью. Причина в том, что в процессе разблокировки и расходования денежных средств некоторые кошельки могут сгенерировать измененный адрес, если расходуется не вся сумма. Кроме того, если компьютер, используемый для подписи транзакции, ненадежен, то вы рискуете открыть секретный ключ. Расходуя сразу всю сумму бумажного кошелька, вы снижаете риск компрометации ключа. Если необходимо потратить меньшую сумму, то следует в той же транзакции переслать все оставшиеся денежные средства в новый бумажный кошелек.

Бумажные кошельки могут иметь различный внешний вид и размеры, а также обладать разнообразными свойствами. Некоторые кошельки создаются как подарки и оформлены в соответствии с временами года, например часто используются темы Рождества и Нового года. Другие создаются специально для хранения в банковских ячейках и сейфах с секретными ключами, скрытыми какими-либо способами, например непрозрачный стирающийся слой (скрэтч-карта) или сгибание бумажного кошелька таким образом, чтобы секретные ключи не были видны, с фиксацией сгиба специальной наклейкой. На рис. 4.10, 4.11 и 4.12 показаны различные примеры бумажных кошельков с возможностями защиты и резервного копирования.

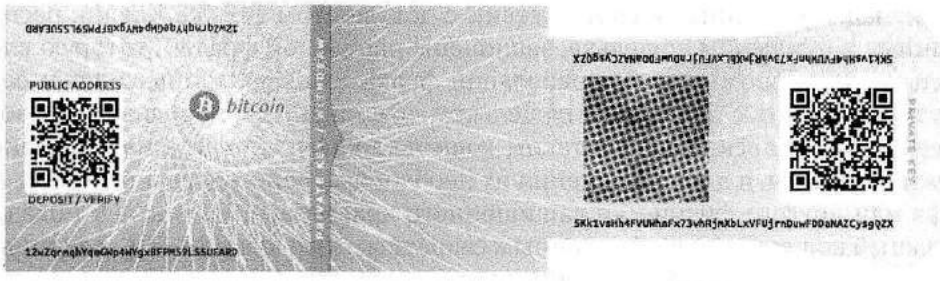


Рис. 4.10 ❖ Пример бумажного кошелька, изготовленного с использованием сайта bitcoinpaperwallet.com с секретным ключом, размещенным на отгибаемом клапане

Рис. 4.11 ❖ Тот же бумажный кошелек bitcoinpaperwallet.com с закрытым и запечатанным клапаном, на котором размещен секретный ключ

Другие варианты дизайна кошельков включают дополнительные копии ключа и адреса в форме отрывного талона, похожего на отрывные талоны пассажирских билетов, которые позволяют хранить несколько копий в различных местах, защищая их от пожаров, затоплений и прочих природных катаклизмов.

Рис. 4.12 ❖ Пример бумажного кошелька с дополнительными копиями ключей на резервном «отрывном талоне»

Глава 5

Кошельки

Слово «кошелек» (wallet) используется для обозначения нескольких различных объектов в биткойн-системе.

На самом верхнем уровне кошелек – это приложение, которое работает как основной пользовательский интерфейс. Кошелек регулирует доступ к деньгам пользователя, управляет ключами и адресами, отслеживает баланс, а также создает и подписывает транзакции.

В более узком смысле с точки зрения программиста слово «кошелек» обозначает структуру данных, используемую для хранения и управления ключами пользователя.

В этой главе мы будем рассматривать кошельки во втором смысле, то есть как контейнеры для секретных ключей, обычно реализованные в форме структурированных файлов или простых баз данных.

Общий обзор технологии кошельков

В этом разделе приводится общий обзор различных технологий, используемых для создания удобных, надежных и универсальных биткойн-кошельков.

Широко распространено ошибочное представление о том, что биткойн-кошельки содержат биткойны. В действительности в кошельке хранятся только ключи. Сами биткойны зафиксированы в структуре данных блокчейна в биткойн-сети. Пользователи управляют биткойнами в этой сети, подписывая транзакции с помощью ключей из своих кошельков. В некотором смысле биткойн-кошелек представляет собой связку ключей или брелок (keychain).

Биткойн-кошельки содержат ключи, а не «монеты». У каждого пользователя есть кошелек с ключами. В действительности кошельки – это брелоки или связки пар секретных/открытых ключей (см. раздел «Секретные и открытые ключи» в главе 4). Пользователи подписывают транзакции с помощью ключей, подтверждая, что именно они владеют выходными данными (биткойнами) конкретной транзакции. Сами биткойны (монеты) хранятся в структуре данных блокчейна в форме выходных данных транзакций (часто обозначаемых аббревиатурами vout или txout).

Существуют два основных типа кошельков, различаемых по следующему признаку: связаны ли хранимые ключи друг с другом или нет.

Первый тип – недетерминированный кошелек (nondeterministic wallet), в котором каждый ключ независимо от других сгенерирован на основе случайного числа. Ключи не связаны друг с другом. Этот тип кошелька также называют JBOK-кошельком по первым буквам фразы «Just a Bunch Of Keys» (Всего лишь связка ключей).

Второй тип – детерминированный кошелек (deterministic wallet), где все ключи произведены от одного главного ключа, называемого источником (seed). В этом типе кошелька все ключи связаны друг с другом и всегда могут быть сгенерированы снова при наличии источника. Существуют различные методики получения ключей от одного источника (key derivation), используемые в детерминированных кошельках. Наиболее часто применяется методика на основе древовидной структуры, по которой создается иерархический детерминированный кошелек (hierarchical deterministic wallet), или HD-кошелек.

Детерминированные кошельки инициализируются значением источника. Для упрощения источники кодируются словами английского языка, называемыми мнемоническими кодовыми словами (mnemonic code words).

В следующих разделах эти технологии будут рассмотрены более подробно.

Недетерминированные кошельки (со случайным выбором ключей)

В первой реализации биткойн-кошелька (теперь она называется Bitcoin Core) сами кошельки представляли собой набор случайно сгенерированных ключей. Например, исходная версия клиента Bitcoin Core при первом запуске предварительно генерирует 100 случайных секретных ключей, затем при необходимости генерирует дополнительные ключи, используя каждый ключ только один раз. Такие кошельки рекомендуется заменять детерминированными версиями, поскольку недетерминированными кошельками очень трудно управлять, создавать резервные копии и выполнять операции импортирования. Основной недостаток случайно выбранных ключей: если генерируется много ключей, то необходимо хранить копии всех ключей, таким образом, приходится слишком часто создавать резервные копии такого кошелька. Каждый ключ требует обязательной резервной копии, иначе можно безвозвратно потерять управляемые им денежные средства, если кошелек станет недоступным. Это напрямую противоречит принципу исключения повторного использования адресов, то есть использования каждого биткойн-адреса только для одной транзакции. Повторное использование адресов снижает уровень секретности, поскольку связывает несколько транзакций и адресов друг с другом. Недетерминированный кошелек Type-0 – неудачный выбор, особенно если нужно избежать повторного использования адресов, так как приходится управлять множеством ключей с необходимостью частого создания резервных копий. В ПО клиента Bitcoin Core включен кошелек Type-0, тем не менее сами разработчики Bitcoin Core не рекомендуют пользоваться этим кошельком. На рис. 5.1 показан

недетерминированный кошелек, содержащий произвольный набор случайно выбранных ключей.

Недетерминированные кошельки рекомендуется использовать только для проведения простых тестов. Они просто слишком сложны для практического применения и резервного копирования. Вместо этого используйте основанный на промышленном стандарте HD-кошелек с мнемоническим источником, резервное копирование которого выполняется намного проще.

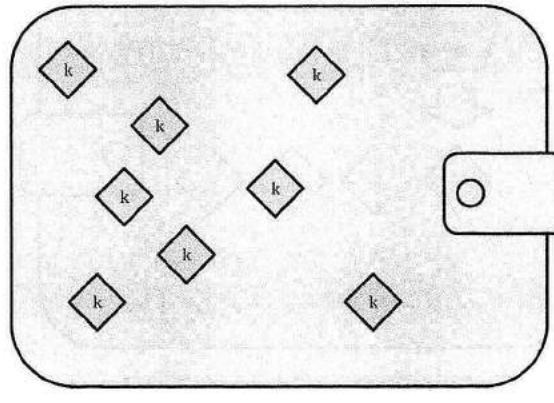


Рис. 5.1 ❖ Недетерминированный кошелек Type-0: набор ключей, сгенерированных случайным образом

Детерминированные кошельки (с источником)

Детерминированные кошельки или кошельки «с источником» (seeded) содержат секретные ключи, произведенные от одного общего источника с помощью односторонней хэш-функции. Источник (seed) – это случайно сгенерированное число, которое объединяется с другими данными, такими как номер индекса или «код цепочки» (chain code) (см. раздел «HD-кошельки (BIP-32/BIP-44)» ниже в этой главе) для генерации производных секретных ключей. В детерминированном кошельке наличия источника достаточно для восстановления всех производных ключей, таким образом, необходимо создать только одну резервную копию сразу после создания такого кошелька. Для экспортирования или импортирования кошелька также достаточно выполнить операцию только с источником, поэтому любые перемещения всех пользовательских ключей между различными версиями кошельков осуществляются очень просто. На рис. 5.2 показана логическая схема детерминированного кошелька.

HD-кошельки (BIP-32/BIP-44)

Детерминированные кошельки были разработаны для упрощения генерации множества ключей, производных от одного источника. Наиболее усовершенствованной формой детерминированных кошельков является HD-кошелек,

определенный в стандарте BIP-32. HD-кошельки содержат ключи, генерируемые в форме древовидной структуры, где ключ-предок может порождать последовательность ключей-потомков, каждый из которых, в свою очередь, может порождать последовательность ключей-потомков следующего «поколения», и так далее до бесконечности. Схема такой древовидной структуры показана на рис. 5.3.

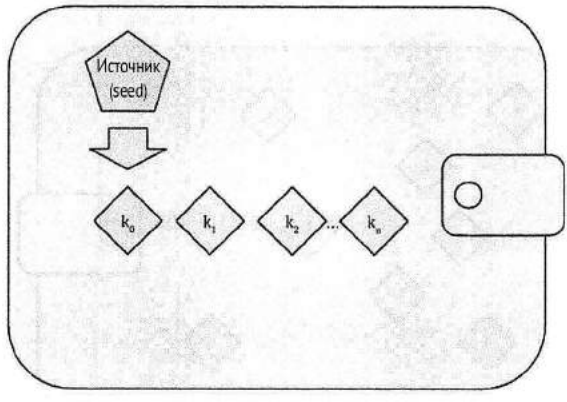


Рис. 5.2 ❖ Детерминированный кошелек Type-1 (с источником): детерминированная последовательность ключей, произведенных от одного источника

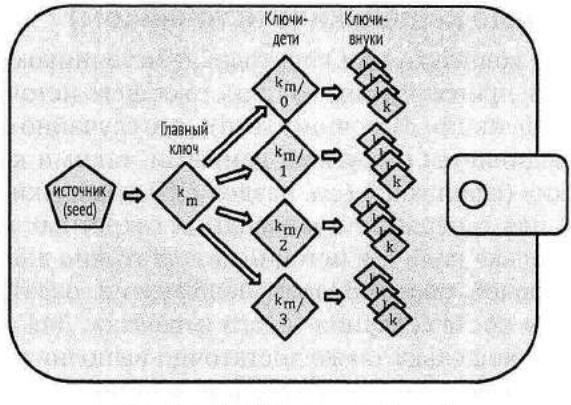


Рис. 5.3 ❖ HD-кошелек Type-2: дерево ключей, генерируемых из одного источника

HD-кошельки обладают двумя основными преимуществами, по сравнению с наборами случайных (недетерминированных) ключей. Во-первых, структура дерева может использоваться как дополнительное организационное средство, например одну из ветвей ключей-потомков можно назначить для приема входящих платежей, другую ветвь – для получения сдачи от исходящих платежей.

Кроме того, ветви ключей можно адаптировать к структуре предприятия, назначая различные ветви для отделов, филиалов, для выполнения особых функций или для специальных категорий счетов.

Второе преимущество HD-кошельков состоит в том, что пользователи могут создавать последовательность открытых ключей без доступа к соответствующим секретным ключам. Это позволяет использовать HD-кошельки на ненадежных серверах или для обеспечения только функции получения средств, генерируя собственный открытый ключ для каждой транзакции. В этом случае открытые ключи не требуется загружать или генерировать заранее, а на сервере не хранятся секретные ключи, которые могут расходовать денежные средства.

Источники и мнемонические коды (BIP-39)

HD-кошельки представляют собой чрезвычайно мощный механизм управления многочисленными ключами и адресами. Такие кошельки еще более полезны, если объединены со стандартизированными методиками создания источников (seeds) из последовательности слов английского языка, которую легко записать, экспортировать и импортировать между кошельками. Такая последовательность называется мнемонической (mnemonic), она определяется стандартом BIP-39. В настоящее время большинство биткойн-кошельков (и кошельков для других криптовалют) использует этот стандарт и получает возможность импорта/экспорта источников для создания резервных копий и восстановления дерева ключей при помощи совместимых и переносимых мнемонических кодов.

Рассмотрим мнемонические коды с практической точки зрения. Какой из приведенных ниже источников проще воспроизвести, записать на бумаге, прочитать без ошибок, экспортировать и импортировать в другой кошелек?

```
0C1E24E5917779D297E14D45F14E1A1A
```

```
army van defense carry jealous true
garbage claim echo media make crunch
```

Оптимальные практические методики технологии кошельков

В технологии биткойн-кошельков уже установлены сформированные общепринятые конкретные промышленные стандарты, обеспечивающие совместимость кошельков, удобство их использования, безопасность и универсальность. Эти общие стандарты перечислены ниже:

- мнемонические кодовые слова – стандарт BIP-39;
- HD-кошельки – стандарт BIP-32;
- многоцелевая структура HD-кошелька – стандарт BIP-43;
- кошельки с поддержкой нескольких валют и нескольких учетных записей – стандарт BIP-44.

Эти стандарты могут измениться или стать устаревшими в результате будущих разработок, но сейчас они формируют комплект взаимосвязанных технологий, которые уже стали стандартом де-факто для биткойн-кошельков.

Перечисленные выше стандарты адаптированы к широкому спектру программных и аппаратных средств для биткойн-кошельков, обеспечивая их совместимость. Пользователь может экспортировать мнемонический код, сгенерированный в одном из кошельков, и импортировать его в другой кошелек, восстановив все транзакции, ключи и адреса.

Вот некоторые примеры программно реализованных кошельков, поддерживающих указанные стандарты (в алфавитном порядке): Breadwallet, Copay, Multibit HD и Mycelium. Примеры аппаратно реализованных кошельков, поддерживающих указанные стандарты (в алфавитном порядке): Keepkey, Ledger и Trezor.

В следующих разделах мы рассмотрим эти технологии более подробно.

Если вы занимаетесь реализацией биткойн-кошелька, то учтите, что он должен быть создан как HD-кошелек с источником (seed), закодированным в форме мнемонической фразы для упрощения резервного копирования. Кроме того, кошелек должен соответствовать стандартам BIP-32, BIP-39, BIP-43 и BIP-44, описанным в предыдущих разделах.

Практическое использование биткойн-кошелька

В разделе «Варианты использования биткойнов, пользователи и их истории» главы 1 мы познакомились с Габриэлем (Gabriel), предприимчивым юношей из Рио-де-Жанейро, организовавшим небольшой веб-магазин, в котором продаются футболки, кофейные кружки и стикеры с символикой биткойна.

Габриэль использует аппаратный кошелек Trezor (рис. 5.4) для безопасного управления своими биткойнами. Trezor – это простое USB-устройство с двумя кнопками для хранения ключей (в форме HD-кошелька) и для подписи транзакций. В кошельках Trezor реализованы все промышленные стандарты, описанные в этой главе, поэтому Габриэль не зависит от какой-либо проприетарной технологии или от единственного решения какого-либо производителя.

Рис. 5.4 ❖ Устройство Trezor:
аппаратный HD-кошелек для биткойнов

Когда Габриэль в первый раз воспользовался кошельком Trezor, устройство сгенерировало мнемонические слова и источник из встроенного аппаратного генератора случайных чисел. На стадии инициализации кошелек выводит на экран пронумерованную последовательность слов поочередно, слово за словом (см. рис. 5.5).

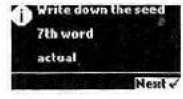


Рис. 5.5 ❖ Trezor выводит на экран одно из мнемонических слов

Записав эту мнемоническую комбинацию, Габриэль создал резервную копию (см. табл. 5.1), которую можно использовать для восстановления при потере или повреждении устройства Trezor. Эту же мнемоническую комбинацию можно применить для восстановления на новом устройстве Trezor или на любом другом совместимом аппаратном или программном кошельке. Отметим, что эта последовательность слов очень важна, поэтому в бумажной форме для резервной копии мнемонической комбинации для каждого слова предназначена пронумерованная свободная ячейка. Габриэль внимательно и аккуратно записал каждое слово в ячейку с соответствующим номером, чтобы сохранить правильный порядок слов в последовательности.

Таблица 5.1. Бумажная форма для резервной копии мнемонической комбинации Габриэля

1.	*army*	**7.**	*garbage*
2.	*van*	**8.**	*claim*
3.	*defence*	**9.**	*echo*
4.	*carry*	**10.**	*media*
5.	*jealous*	**11.**	*make*
6.	*true*	**12.**	*crunch*

В табл. 5.1 показана мнемоническая комбинация из 12 слов как упрощенный учебный пример. В действительности большинство аппаратных кошельков генерирует более безопасную мнемоническую комбинацию из 24 слов. Мнемоническая последовательность всегда используется абсолютно одинаково независимо от ее длины.

Для первой версии своего веб-магазина Габриэль использовал один биткойн-адрес, сгенерированный устройством Trezor. Этот адрес используется всеми покупателями для всех заказов. Как мы увидим в дальнейшем, такой подход имеет некоторые недостатки и может быть улучшен с помощью HD-кошелька.

Подробности технологии кошельков

А теперь более подробно рассмотрим каждый из важных промышленных стандартов, используемых многими биткойн-кошельками.

Мнемонические кодовые слова (BIP-39)

Мнемонические кодовые слова – это последовательности слов, которые представляют (кодируют) случайное число, используемое как источник для формирования детерминированного кошелька. Этой последовательности слов достаточно, чтобы воссоздать источник, а из него – кошелек и все порожденные ключи. Приложение кошелька, реализующее детерминированный тип кошелька с мнемоническими словами, предъявляет пользователю последовательность длиной от 12 до 24 слов, когда кошелек используется в первый раз. Эта последовательность слов является резервной копией данного конкретного кошелька и может применяться для восстановления и воссоздания всех ключей в том же кошельке или в любом совместимом приложении кошелька. Мнемонические слова существенно упрощают процедуру резервного копирования кошельков, потому что пользователю гораздо легче прочитать их и правильно записать, по сравнению со случайной последовательностью чисел.

Мнемонические слова часто путают с «умственными кошельками» (brainwallets). Это не одно и то же. Главное различие заключается в том, что для умственного кошелька последовательность слов выбирает сам пользователь, а мнемонические слова создаются случайным образом приложением кошелька и предлагаются пользователю. Это важное различие обеспечивает гораздо большую надёжность мнемонических слов, поскольку люди – плохие источники случайности.

Мнемонические коды определены в документе BIP-39 (см. приложение B). Отметим, что BIP-39 – это одна из реализаций стандарта мнемонических кодов. Существует другой стандарт с другим набором слов, используемый кошельком Electrum и сформированный раньше, чем BIP-39. Стандарт BIP-39 был предложен компанией, производящей аппаратные кошельки Trezor, и он не совместим с реализацией Electrum. Тем не менее в настоящее время стандарт BIP-39 получил широкую поддержку производителей, имеются десятки его реализаций, поэтому следует считать его промышленным стандартом де-факто.

Стандарт BIP-39 определяет создание мнемонических кодов и источника (seed) в виде процедуры из девяти шагов, описанной ниже. Для лучшего понимания процедура разделена на две части: шаги с 1 по 6 показаны в разделе «Генерация мнемонических слов», а шаги с 7 по 9 – в разделе «От мнемонической последовательности к источнику (seed)».

Генерация мнемонических слов

Мнемонические слова генерируются автоматически кошельком с использованием стандартного процесса, определенного в документе BIP-39. Кошелек

начинает процедуру с создания источника энтропии (случайности), добавляет контрольную сумму, затем выполняет отображение случайной последовательности на список слов:

1. Создание случайной последовательности (источника энтропии) длиной от 128 до 256 битов.
2. Создание контрольной суммы этой случайной последовательности, вычисляемой как первые (длина_последовательности/32) битов ее хэш-значения SHA256.
3. Добавление контрольной суммы в конец случайной последовательности.
4. Разделение последовательности на секции длиной 11 битов каждая.
5. Установление соответствия каждому 11-битовому значению слова из предварительно определенного словаря, состоящего из 2048 слов.
6. Полученная последовательность слов является мнемоническим кодом.

На рис. 5.6 показано, как энтропия (случайность) используется для генерации мнемонических слов.

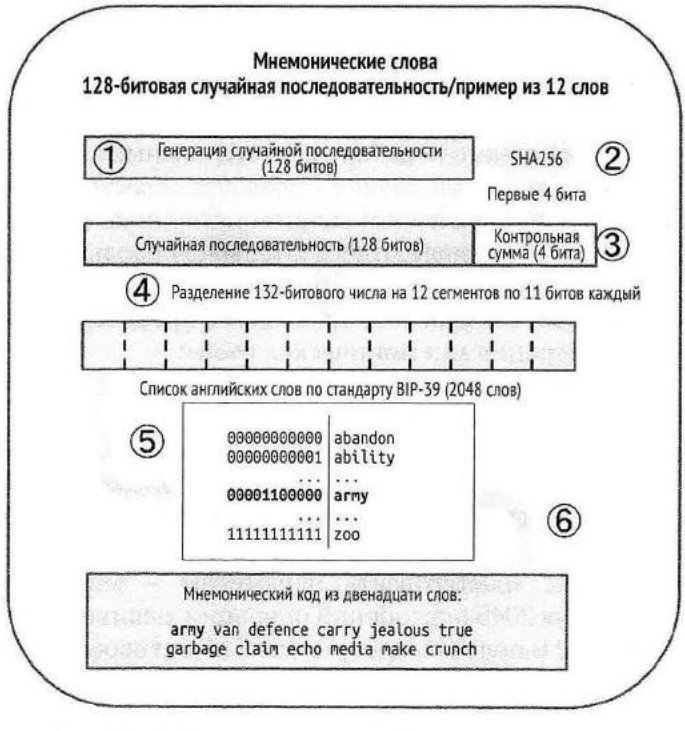


Рис. 5.6 ❖ Генерация случайной последовательности и кодирование мнемонических слов

В табл. 5.2 показано соотношение между размером случайной последовательности и длиной мнемонических кодов в словах.

Таблица 5.2. Мнемонические коды: размер случайной последовательности и длина списка слов

Случайная последовательность (длина в битах)	Контрольная сумма (длина в битах)	Случайная последовательность + контрольная сумма (длина в битах)	Длина мнемонического кода (в словах)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

От мнемонических слов к источнику (seed)

Мнемонические слова представляют случайную последовательность длиной от 128 до 256 битов. Далее эта случайная последовательность используется для формирования более длинного (512 битов) источника (seed) с помощью специальной функции «растягивания» ключа PBKDF2. Полученный источник затем используется для создания детерминированного кошелька и для порождения ключей.

Функция «растягивания» ключа принимает два параметра: мнемонический код и соль (или модификатор; salt). Назначение соли (модификатора) в этой функции – затруднить построение таблицы поиска для осуществления атаки грубой силой (методом полного перебора всех возможных ключей). В стандарте BIP-39 соль имеет и другое назначение – позволяет ввести парольную фразу, служащую дополнительным фактором защиты источника. Более подробно об этом будет сказано ниже, в разделе «Необязательная парольная фраза по стандарту BIP-39» текущей главы.

Процесс, описываемый шагами 7–9, продолжает процесс, описанный в предыдущем разделе «Генерация мнемонических слов»:

7. Первый параметр для функции PBKDF2 – мнемонический код, полученный на шаге 6.
8. Второй параметр для функции PBKDF2 – соль (salt), или модификатор. Соль формируется из строковой константы `"mnemonic"`, объединяемой с необязательной строкой парольной фразы, предоставляемой пользователем.
9. Функция PBKDF2 «растягивает» параметры – мнемонический код и соль, – используя 2048 повторений операции хэширования по алгоритму HMAC-SHA512 и получая в результате 512-битовое значение, которое представляет собой источник (seed).

На рис. 5.7 показано использование мнемонических кодов для генерации источника.

Рис. 5.7 ❖ От мнемонического кода к источнику

Функция «растягивания» ключа с 2048 повторениями операции хэширования является весьма эффективной защитой от атак с применением грубой силы (полного перебора) на мнемонические коды или на парольную фразу. Затраты (вычислительные) становятся чрезвычайно большими при попытках перебора всех возможных комбинаций из нескольких тысяч парольных фраз и мнемонических кодов, тогда как количество возможных генерируемых источников огромно (2^{512}).

В табл. 5.3, 5.4 и 5.5 показаны примеры мнемонических кодов и созданных с их помощью источников (без парольных фраз и с парольной фразой).

Таблица 5.3. 128-битовый случайный мнемонический код, без парольной фразы, итоговый источник

Исходная случайная последовательность (128 битов)
`0c1e24e5917779d297e14d45f14e1a1a`
Мнемонический код (12 слов)
`army van defense carry jealous true garbage claim echo media make crunch`
Парольная фраза
`(нет)`
Источник (512 битов)
`5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891 bfaba16ed27a813ceed498804c0570`

Таблица 5.4. 128-битовый случайный мнемонический код, с парольной фразой, итоговый источник

Исходная случайная последовательность (128 битов)
0c1e24e5917779d297e14d45f14e1a1a
Мнемонический код (12 слов)
army van defense carry jealous true garbage claim echo media make crunch
Парольная фраза
SuperDuperSecret
Источник (512 битов)
3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0715861dc8a18358f80b79d49acf64142ae57037d1d54

Таблица 5.5. 256-битовый случайный мнемонический код, без парольной фразы, итоговый источник

Исходная случайная последовательность (256 битов)
2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Мнемонический код (24 слова)
cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Парольная фраза
(нет)
Источник (512 битов)
3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e55f1e0deaa082df8d487381379df848a6ad7e98798404

Необязательная парольная фраза по стандарту BIP-39

Стандарт BIP-39 разрешает использование необязательной парольной фразы в процессе формирования источника. Если парольная фраза не задана, то мнемонический код «растягивается» с использованием соли, состоящей из строковой константы "mnemonic". В результате получается уникальный 512-битовый источник из любого заданного мнемонического кода. Если парольная фраза задана, то функция создает совершенно другой источник из того же мнемонического кода. Таким образом, при одном и том же мнемоническом коде каждая возможная парольная фраза приводит к созданию другого источника. В сущности, здесь не может быть «неправильных» парольных фраз. Все парольные фразы являются допустимыми, и все они приводят к созданию различных источников, тем самым образуя огромный набор потенциально возможных неинициализированных кошельков. Этот набор настолько велик (2^{512}), что практически нет никакой возможности найти полным перебором или случайно угадать реально используемый кошелек.

По стандарту BIP-39 нет «неправильных» парольных фраз. Любая парольная фраза соответствует некоторому кошельку. Если кошелек до этого не использовался, то он будет пустым.

Дополнительная парольная фраза создает две важные функциональные характеристики:

- второй фактор (также требует запоминания), при введении которого мнемонические коды сами по себе становятся бесполезными. Это защищает резервные копии мнемонических кодов даже в случае их похищения;
- форма правдоподобной маскировки, или «ложный кошелек», когда конкретная выбранная парольная фраза указывает путь к кошельку с чрезвычайно малой денежной суммой, тем самым отвлекая внимание атакующего от настоящего кошелька с основными денежными средствами.

Тем не менее важно отметить, что использование парольной фразы также создает риск потери:

- если владелец кошелька недееспособен или мертв и никто другой не знает парольной фразы, то источник бесполезен и все денежные средства, хранящиеся в этом кошельке, потеряны навсегда;
- и напротив, если владелец хранит резервную копию парольной фразы в том же месте, где хранится источник, то это полностью лишает смысла второй фактор защиты.

Парольные фразы весьма полезны, но их следует использовать только при условии тщательного планирования процессов резервного копирования и восстановления с учетом физических возможностей владельца, чтобы обеспечить восстановление права владения денежными средствами им или членами его семьи.

Работа с мнемоническими кодами

Стандарт BIP-39 реализован как библиотека на многих языках программирования:

python-mnemonic (https://github.com/trezor/python-mnemonic)
Эталонная реализация стандарта на языке Python от группы SatoshiLabs, которая предложила стандарт BIP-39.

bitcoinjs/bip39 (https://github.com/bitcoinjs/bip39)
Реализация стандарта BIP-39 как части широко известной программной среды bitcoinJS на языке JavaScript.

libbitcoin/mnemonic (https://github.com/libbitcoin/libbitcoin/blob/master/src/wallet/mnemonic.cpp)
Реализация стандарта BIP-39 как части широко известной программной среды Libbitcoin на языке C++.

Существует также генератор BIP-39, реализованный в форме независимой веб-страницы, чрезвычайно удобной для тестирования и различных экспериментов. На рис. 5.8 показана независимая веб-страница, на которой можно генерировать мнемонические коды, источники и усовершенствованные секретные ключи.

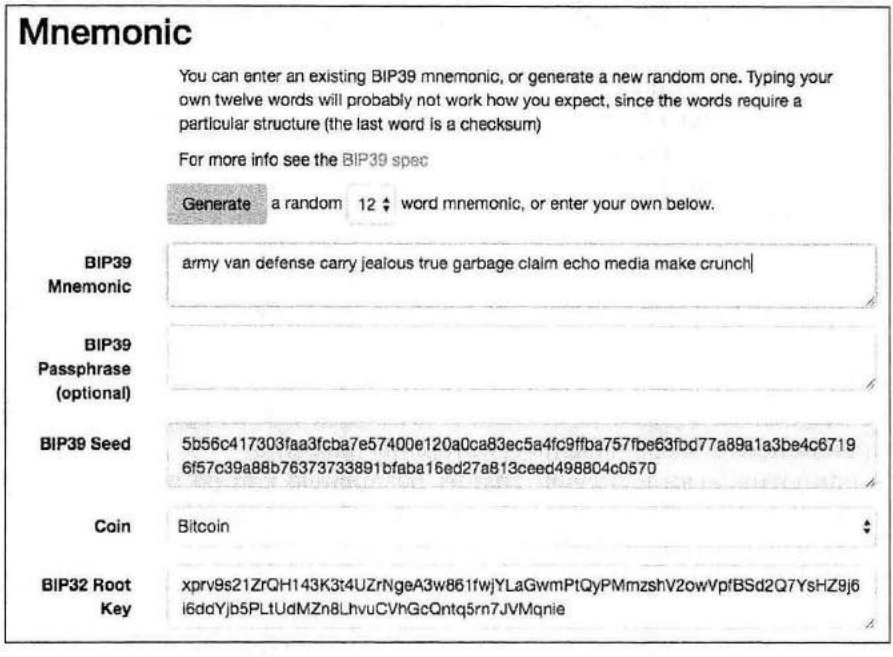


Рис. 5.8 ❖ Генератор BIP-39 как независимая веб-страница

С этой страницей можно работать в режиме офлайн в локальном браузере или получить к ней доступ в режиме онлайн (https://dcpos.github.io/bip39/).

Создание HD-кошелька из источника

HD-кошельки создаются из одного корневого источника (root seed), являющегося 128-, 256- или 512-битовым случайным числом. В большинстве случаев этот источник генерируется из мнемонических кодов, как описано в предыдущем разделе.

Каждый ключ в HD-кошельке детерминистически выводится из этого корневого источника, что дает возможность воссоздавать весь HD-кошелек из одного лишь источника в любом совместимом HD-кошельке. Это существенно упрощает резервное копирование, восстановление, экспорт и импорт HD-кошельков, содержащих тысячи или даже миллионы ключей, просто передавая в нужное место только мнемоническую последовательность, из которой генерируется корневой источник.

Процесс создания главных ключей и главной кодовой цепочки кодовых слов для HD-кошелька показан на рис. 5.9.

Корневой источник представляет собой входные данные для алгоритма HMAC-SHA512, а полученное в результате хэш-значение используется для создания главного секретного ключа (m) и главной цепочки кодовых слов (c).

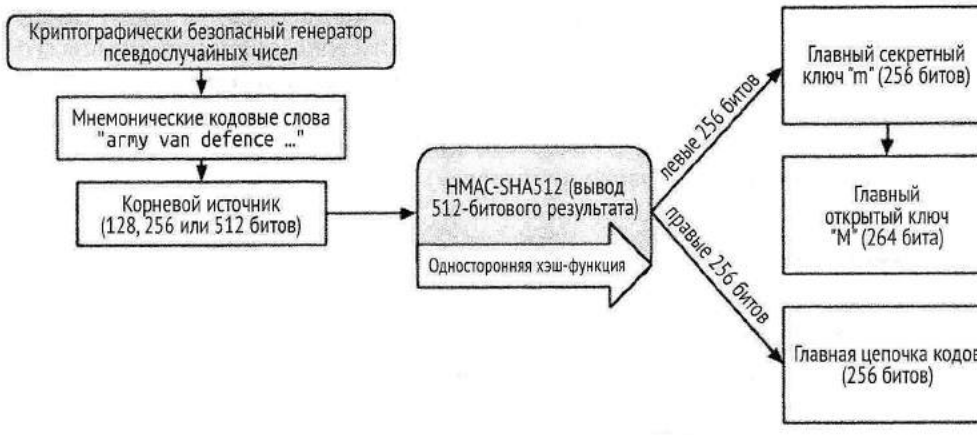


Рис. 5.9 ❖ Создание главных ключей и цепочки кодовых слов из корневого источника

Затем из главного секретного ключа (m) генерируется соответствующий главный открытый ключ (M) с использованием обычного процесса умножения на эллиптической кривой `m* G`, который был показан в разделе «Открытые ключи» главы 4.

Цепочка кодов (c) используется для ввода случайности (энтропии) в функцию, создающую ключи-потомки из ключей-родителей, как мы увидим в следующем разделе.

Генерация секретного ключа-потомка

В HD-кошельках используется функция child key derivation (CKD) для генерации ключей-потомков из ключей-родителей.

Функции генерации ключей-потомков основаны на односторонней хэш-функции, которая объединяет:

- секретный или открытый ключ-родитель (несжатый ECDSA-ключ);
- источник (seed), называемый цепочкой кодов (chain code) (256 битов);
- номер индекса (32 бита).

Цепочка кодов (chain code) используется для ввода детерминированных случайных данных в процесс, для того чтобы знание номера индекса и ключа-потомка стало недостаточным условием для генерации других ключей-потомков. Знание любого ключа-потомка не дает возможности узнать значения его собратьев, если неизвестна цепочка кодов. Начальный источник цепочки кодов (в корне дерева) создается из корневого источника, тогда как все последующие цепочки кодов генерируются из каждой родительской цепочки кодов.

Эти три элемента (ключ-родитель, цепочка кодов и индекс) объединяются и хэшируются для генерации ключей-потомков, как показано ниже.

Открытый ключ-родитель, цепочка кодов и номер индекса объединяются и хэшируются с помощью алгоритма HMAC-SHA512 для получения 512-би-

тового хэш-значения. Это значение разделяется на две равные части по 256 битов. 256 битов правой половины полученного хэш-значения становятся цепочкой кодов для потомка. 256 битов левой половины и номер индекса добавляются к секретному ключу-родителю для создания секретного ключа-потомка. На рис. 5.10 показана схема этого процесса с номером индекса 0 для получения потомка zero (первого по индексу) данного родителя.

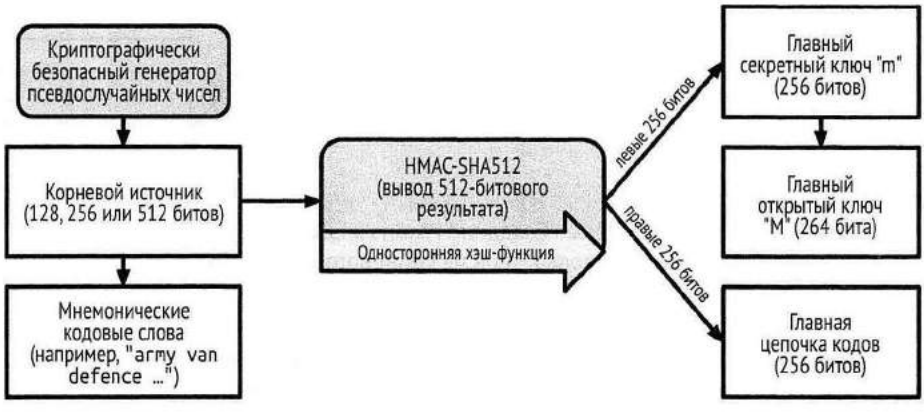


Рис. 5.10 ❖ Увеличение размера секретного ключа-родителя для создания секретного ключа-потомка

Изменение индекса позволяет изменять значение ключа-родителя с помощью увеличения его длины и последовательно создавать других потомков, например Child 0, Child 1, Child 2 и т. д. Каждый ключ-родитель может иметь 2 147 483 647 (2^{31}) потомков (2^{31} – это половина всего диапазона 2^{32}; доступна только половина диапазона, потому что другая половина предназначена для особенного типа генерации, который мы рассмотрим немного позже в этой главе).

Повторяя описанный выше процесс на более низких уровнях дерева, можно превращать каждого потомка в родителя и создавать его собственных потомков, продолжая процедуры генерации до бесконечности.

Использование производных ключей-потомков

Секретные ключи-потомки ничем не отличаются от недетерминированных (случайных) ключей. Поскольку функция генерации является односторонней (однонаправленной), ключ-потомок невозможно использовать для вычисления ключа-родителя. Ключ-потомок также не позволяет определить значение какого-либо ключа-собрата. Если известен n-й потомок, то вы не сможете узнать значения его собратьев, то есть ни (n–1)-го, ни (n+1)-го, ни какого-либо другого потомка, являющегося частью данной последовательности. Только при наличии ключа-родителя и цепочки кодов можно воспроизвести всех потомков. Без цепочки кодов-потомков ключ-потомок невозможно использо-

вать для генерации следующего поколения потомков. Для начала новой ветви и создания нового поколения необходимы и секретный ключ-потомок, и цепочка кодов-потомков.

А для чего можно использовать секретный ключ-потомок сам по себе? Для генерации открытого ключа и биткойн-адреса. После этого его можно применять для подписи транзакций, расходующих средства, поступающие на этот адрес.

Секретный ключ-потомок, соответствующий открытый ключ и биткойн-адрес неотличимы от ключей и адресов, созданных случайным образом. Тот факт, что они являются частью некоторой последовательности, неизвестен за пределами функции HD-кошелька, с помощью которой они были созданы. После создания они работают точно так же, как обычные ключи и адреса.

Расширяемые ключи

Как мы видели выше, функция порождения ключей может использоваться для создания потомков на любом уровне дерева, принимая три элемента входных данных: ключ, цепочку кодов и номер индекса требуемого потомка. Двумя особенно важными элементами являются ключ и цепочка кодов, а их объединение называется расширяемым ключом (extended key). Именно «расширяемым», поскольку английский термин «extended key» можно трактовать еще и как «extensible key», потому что такой ключ можно использовать для порождения потомков.

Расширяемые ключи хранятся и выводятся в виде, представляющем собой простое объединение 256-битового ключа и 256-битовой цепочки кодов в 512-битовую последовательность. Существуют два типа расширяемых ключей. Расширяемый секретный ключ – это объединение секретного ключа и цепочки кодов, его можно использовать для генерации секретных ключей-потомков (а из них – открытых ключей-потомков). Расширяемый открытый ключ – это исходный открытый ключ и цепочка кодов, его можно использовать для создания открытых (только открытых) ключей-потомков, как описано в разделе «Генерация открытого ключа» главы 4.

Можно считать расширяемый ключ корнем ветви в структуре дерева HD-кошелька. Из корня ветви можно сгенерировать всю остальную ветвь. С помощью расширяемого секретного ключа можно создать полноценную ветвь (со всеми компонентами), в то время как расширяемый открытый ключ позволяет создать только ветвь открытых ключей.

Расширяемый ключ состоит из секретного или открытого ключа и цепочки кодов. Расширяемый ключ позволяет создавать ключи-потомки, формируя собственные ветви в структуре дерева. Опубликование расширяемого ключа предоставляет доступ ко всей соответствующей ветви.

Расширяемые ключи кодируются с помощью Base58Check для упрощения экспорта и импорта между различными кошельками, совместимыми по стандарту BIP-32. В кодировке Base58Check используется специальный но-

мер версии для расширяемых ключей, который дает префикс "xprv" или "xpub" в символах Base58, что позволяет легко распознавать этот тип ключей. Так как расширяемые ключи имеют длину 512 или 513 битов, они намного длиннее, чем строки в кодировке Base58Check, которые мы видели ранее.

Вот пример расширяемого секретного ключа в кодировке Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6Hx4t
ws2six3b9c
```

Ниже показан соответствующий расширяемый открытый ключ в кодировке Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMstweyLXhRgPxdp
14sk9tJPW9
```

Генерация открытого ключа-потомка

Как было отмечено выше, весьма полезной характеристикой HD-кошельков является их способность генерировать открытые ключи-потомки из открытых ключей-родителей без обязательного применения секретных ключей. Это дает два способа создания открытого ключа-потомка: из секретного ключа-потомка или непосредственно из открытого ключа-родителя.

Таким образом, расширяемый открытый ключ может применяться для порождения всех открытых (и только открытых) ключей в данной ветви структуры дерева HD-кошелька.

Этот упрощенный способ можно использовать для создания хорошо защищенных вариантов развертывания с одними только открытыми ключами, когда сервер или приложение располагает лишь копией расширяемых открытых ключей без каких бы то ни было секретных ключей. При таком типе развертывания можно порождать бесконечное число открытых ключей и биткойн-адресов, но невозможно тратить денежные средства, передаваемые на эти адреса. Одновременно на другом, более защищенном сервере с помощью расширяемого секретного ключа можно сгенерировать все соответствующие секретные ключи-потомки для подписи транзакций, расходующих средства с вышеуказанных адресов.

Одной из широко известных практических реализаций этого решения является установка расширяемого открытого ключа на веб-сервер, обслуживающий приложение электронной коммерции. Такой веб-сервер может применять функцию генерации открытых ключей-потомков для создания нового биткойн-адреса для каждой транзакции (например, для «корзины с покупками» каждого клиента). На этом веб-сервере нет ни одного секретного ключа, который подвергался бы риску быть похищенным. Без использования HD-кошельков единственным способом реализации подобного решения являлась бы генерация тысяч биткойн-адресов на отдельном хорошо защищенном сервере с последующей передачей их на сервер электронной коммерции. Такой подход весьма затруднителен и требует постоянного технического сопровождения для гарантии того, что сервер электронной коммерции не «израсходует» все ключи.

Другое широко распространенное применение этого решения – холодное хранение (cold storage) или аппаратные кошельки (hardware wallets). В этом случае расширяемый секретный ключ может храниться в бумажном кошельке или на аппаратном устройстве (таком как аппаратный кошелек Trezor), в то время как расширяемый открытый ключ можно хранить в режиме онлайн. Пользователь имеет возможность создавать адреса «для получения» по собственному желанию, при этом секретные ключи надежно сохранены в режиме офлайн. Для расходования денежных средств пользователь может применять расширяемый секретный ключ на биткойн-клиенте для подписи в режиме офлайн или подписывать транзакции на аппаратном кошельке (например, Trezor). На рис. 5.11 показан механизм использования расширяемого открытого ключа-родителя для генерации открытых ключей-потомков.

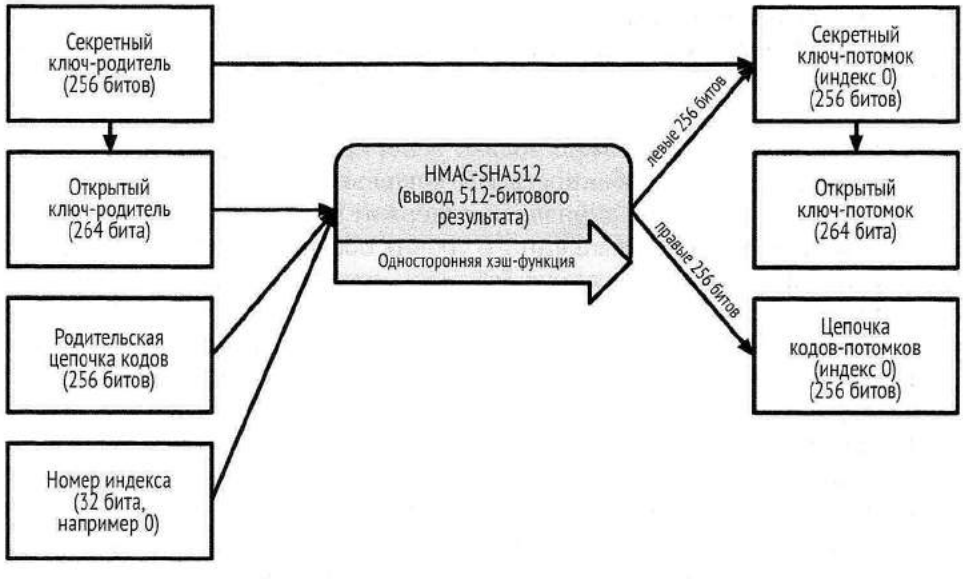


Рис. 5.11 ❖ Использование расширяемого открытого ключа-родителя для создания открытых ключей-потомков

Использование расширяемого открытого ключа в веб-магазине

Рассмотрим подробнее практическое применение HD-кошельков на примере продолжения истории веб-магазина Габриэля.

Сначала Габриэль создал свой веб-магазин как хобби на основе простой страницы, управляемой механизмом Wordpress. Сайт магазина состоял из нескольких страниц и формы заказа с одним биткойн-адресом.

Габриэль использовал первый биткойн-адрес, сгенерированный его устройством Trezor, как основной биткойн-адрес своего магазина. Таким образом, все входящие платежи должны были приходить на адрес, управляемый аппаратным кошельком Trezor.

Покупатели должны были подтверждать заказ в специальной форме и посылать оплату на биткойн-адрес, опубликованный Габриэлем, при этом автоматически формировалось сообщение электронной почты, содержащее подробности заказа и отправляемое Габриэлю для обработки. При нескольких заказах в неделю эта система работала достаточно хорошо.

Но постепенно этот маленький веб-магазин становился все более популярным и успешным и привлекал все большее количество клиентов из местного сообщества. Вскоре Габриэль испытал серьезные затруднения. При оплате всех заказов на один адрес становилось все труднее правильно сопоставлять заказы и транзакции, особенно когда несколько заказов на одинаковое количество товара приходило практически одновременно.

HD-кошелек Габриэля предлагает гораздо лучшее решение благодаря возможности генерировать открытые ключи-потомки без знания секретных ключей. Габриэль может загрузить на свой веб-сайт расширяемый открытый ключ (xpub), который будет использоваться для генерации неповторяющихся адресов для каждого отдельного заказа. Габриэль может расходовать денежные средства со своего аппаратного кошелька Trezor, но с помощью xpub-ключа, загруженного на веб-сайт, можно только генерировать адреса и принимать денежные средства. Эта особенность HD-кошельков является великолепной функцией защиты. Веб-сайт Габриэля не содержит никаких секретных ключей, следовательно, не требует повышенного уровня обеспечения безопасности.

Для экспортирования xpub-ключа Габриэль использует программное обеспечение веб-сайта в сочетании с аппаратным кошельком Trezor. Устройство Trezor необходимо подключить к сетевому узлу, чтобы выполнить экспортирование открытых ключей. Отметим, что аппаратные кошельки никогда не экспортируют секретные ключи – они всегда остаются только на данном устройстве. На рис. 5.12 показан веб-интерфейс, которым пользуется Габриэль для экспорта xpub-ключа.

Габриэль копирует xpub-ключ в программное биткойн-приложение для своего веб-магазина. Затем использует Mycelium Gear – подключаемый модуль (plugin) с открытым исходным кодом, предназначенный не только для веб-магазинов, но и для различных операций на разнообразных платформах веб-хостинга и управления содержимым (контентом). Mycelium Gear использует xpub-ключ для генерации неповторяющегося адреса для каждой операции покупки.

Более безопасная генерация ключей-потомков

Возможность порождения ветви открытых ключей из xpub-ключа очень полезна, но связана с потенциальным риском. Доступ к xpub-ключу не открывает доступа к секретным ключам-потомкам. Но поскольку xpub-ключ содержит цепочку кодов, при получении секретного ключа-потомка каким-либо образом можно воспользоваться им в сочетании с этой цепочкой кодов, чтобы сгенерировать все прочие секретные ключи-потомки. Единственный секретный ключ-потомок, ставший известным злоумышленнику, вместе с родительской

цепочкой кодов открывает доступ ко всем секретным ключам всех потомков. Хуже того, секретный ключ-потомок в сочетании с родительской цепочкой кодов может быть использован для вычисления секретного ключа-родителя.

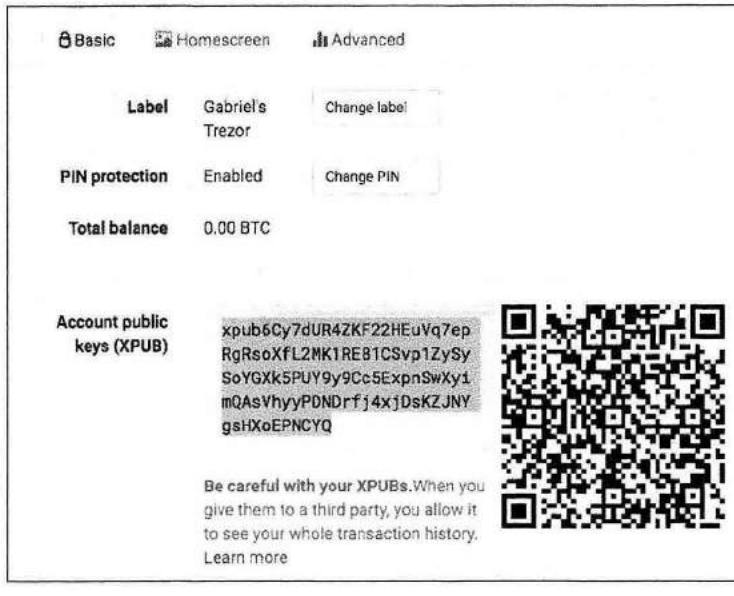


Рис. 5.12 ❖ Экспортирование xpub-ключа из аппаратного кошелька Trezor

Для противодействия этой опасности HD-кошельки используют альтернативную функцию генерации, называемую защищенной генерацией (hardened derivation), которая «разрушает» взаимосвязь между открытым ключом-родителем и цепочкой кодов потомка. Функция защищенной генерации вместо открытого ключа-родителя принимает секретный ключ-родитель для порождения цепочки кодов потомка. Это создает своего рода «защитный барьер» (firewall) в последовательности родитель–потомок с цепочкой кодов, которую невозможно использовать для компрометации секретного ключа-родителя или секретного ключа-собрата. Функция защищенной генерации выглядит почти одинаковой с функцией обычной генерации секретного ключа-потомка, с тем лишь исключением, что вместо открытого ключа-родителя используется секретный ключ-родитель в качестве входных данных для функции хэширования, как показано на схеме рис. 5.13.

При использовании функции защищенной генерации полученный секретный ключ-потомок и цепочка кодов полностью отличаются от ключа-потомка и цепочки кодов, вычисленных обычной функцией генерации. Сгенерированную «ветвь» ключей можно использовать для порождения расширяемых открытых ключей, вполне безопасных, так как цепочку кодов, которую они

содержат, невозможно применить для вычисления каких-либо секретных ключей. Таким образом, функцию защищенной генерации следует использовать для создания «разрыва» в структуре дерева на уровне, в котором применяются расширяемые открытые ключи.

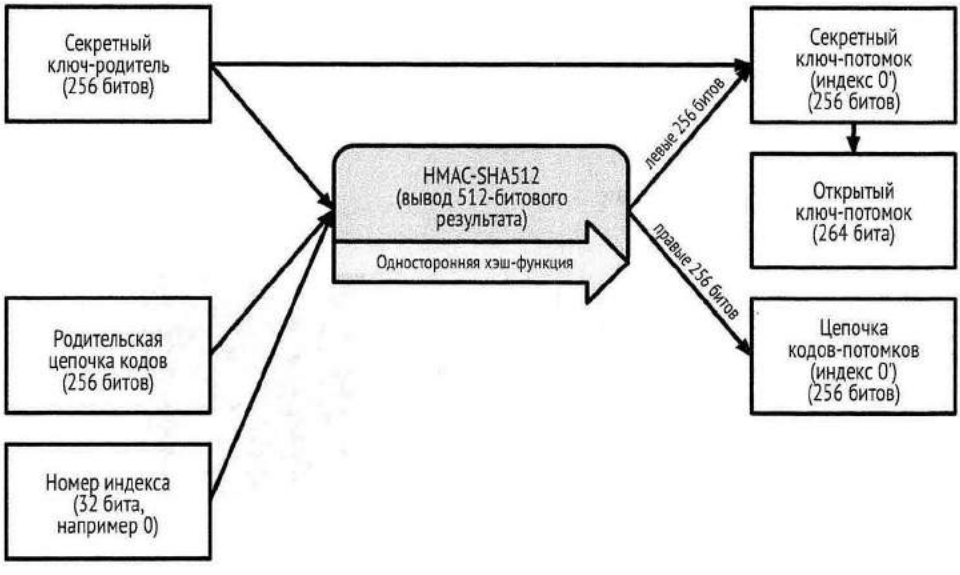


Рис. 5.13 ❖ Функция защищенной генерации ключа-потомка; здесь не используется открытый ключ-родитель

Проще говоря, если необходимо воспользоваться преимуществами xpub-ключа для генерации ветвей открытых ключей без потенциальной угрозы утечки цепочки кодов, рекомендуется порождать потомков от защищенного (hardened) родителя, а не от обычного. Наилучшая практическая реализация: потомки уровня 1 от главных ключей всегда генерируются с помощью функции защищенной генерации, чтобы предотвратить компрометацию главных ключей.

Номера индекса для обычной и защищенной генерации

Номер индекса, используемый в функции генерации, представляет собой 32-битовое число. Чтобы проще было различать ключи, созданные обычной функцией генерации, и ключи, созданные функцией защищенной генерации, весь диапазон индексов разделяется на две части. Номера индекса от 0 до $2^{31}-1$ (от 0x0 до 0x7FFFFFFF) используются только при обычной генерации. Номера индекса от 2^{31} до $2^{32}-1$ (от 0x80000000 до 0xFFFFFFFF) используются только при защищенной генерации. Таким образом, если номер индекса меньше 2^{31}, то потомок «обычный», а если номер индекса больше или равен 2^{31}, то потомок «защищенный».

Чтобы сделать номера индекса более простыми для чтения и восприятия при выводе, номер индекса защищенного потомка всегда выводится с начальным нулем, но этот нуль сопровождается символом одиночного штриха. Таким образом, первый обычный ключ-потомок выводится как 0, в то время как первый защищенный потомок (индекс 0x80000000) выводится как 0'. Соблюдая последовательность, второй защищенный ключ должен иметь индекс 0x80000001 и выводиться как 1' и т. д. Когда вы видите HD-кошелек с индексом i', это означает номер индекса 2^{31}+i.

Идентификатор (путь) ключей в HD-кошельке

Ключи в HD-кошельке идентифицируются в соответствии с «путевым» (path) соглашением по именованию, по которому каждый уровень дерева отделяется символом слэша (/) (см. табл. 5.6). Секретные ключи, порожденные от главного секретного ключа, начинаются с символа «m». Открытые ключи, порожденные от главного открытого ключа, начинаются с символа «M». Таким образом, первый секретный ключ-потомок главного секретного ключа получает имя m/0. Первый открытый ключ-потомок именуется M/0. Второй потомок из следующего поколения от первого потомка называется m/0/1 и т. д.

Такая «родословная» ключа читается справа налево до тех пор, пока не встретится обозначение главного ключа, от которого была выполнена генерация. Например, идентификатор m/x/y/z описывает ключ, являющийся z-м потомком ключа m/x/y, который, в свою очередь, является y-м потомком ключа m/x – x-го потомка главного ключа m.

Таблица 5.6. Примеры путевых имен в HD-кошельке

Путь в HD-кошельке	Описание ключа
m/0	Первый (0) секретный ключ-потомок от главного секретного ключа (m)
m/0/0	Первый секретный ключ-потомок следующего поколения от первого секретного ключа-потомка (m/0)
m/0'/0	Первый обычный потомок следующего поколения от первого *защищенного* секретного ключа-потомка (m/0')
m/1/0	Первый секретный ключ-потомок следующего поколения от второго секретного ключа-потомка (m/1)
M/23/17/0/0	Первый открытый ключ-потомок четвертого поколения от первого открытого ключа-потомка третьего поколения от 18-го потомка второго поколения от 24-го потомка первого поколения

Перемещение по структуре дерева HD-кошелька

Древовидная структура HD-кошелька обеспечивает невероятную гибкость и универсальность. Каждый расширяемый ключ-родитель может иметь 4 миллиарда потомков: 2 миллиарда обычных и 2 миллиарда защищенных (hardened) потомков. Каждый из этих потомков может также иметь 4 миллиарда собственных потомков и т. д. Дерево может быть настолько глубоким, насколько вам нужно, с бесконечным количеством поколений. Но подобная гибкость приводит к затруднениям при необходимости перемещения по такому бес-

конечному дереву. Особенно трудно передавать HD-кошельки между различными реализациями, потому что возможности внутреннего устройства ветвей и подветвей бесконечны.

Два документа BIP предлагают решения по устранению сложности посредством создания некоторых предполагаемых стандартов для структуры деревьев в HD-кошельках. Документ BIP-43 предлагает использовать первый защищенный (hardened) индекс потомка как особый идентификатор, который является признаком «цели» конкретной древовидной структуры. По стандарту BIP-43 HD-кошелек должен использовать только одну ветвь дерева первого уровня (level-1) с номером индекса, идентифицирующим структуру и пространство имен остальной части дерева и определяющим его цель. Например, HD-кошелек, использующий только ветвь m/i'/, предназначен исключительно для обозначения конкретной заданной цели, которая идентифицируется по номеру индекса i.

Расширяя эту спецификацию, документ BIP-44 предлагает структуру со многими учетными записями как «цель» номер 44' по стандарту BIP-43. Все HD-кошельки, имеющие структуру по стандарту BIP-44, идентифицируются по следующему признаку: они используют только одну ветвь дерева – m/44'/.

В документе BIP-44 определяется структура, состоящая из пяти предопределенных уровней дерева:

```
m / purpose' / coin_type' / account' / change / address_index
```

Для первого уровня purpose всегда установлено значение 44'. Второй уровень coin_type определяет тип денежной единицы криптовалюты, позволяя создавать многовалютные HD-кошельки, где каждой денежной единице отведена собственная подветвь дерева, начиная со второго уровня. В настоящее время доступны три обозначения криптовалют: m/44'/0' для Bitcoin, m/44'/1' для Bitcoin Testnet и m/44'/2' для Litecoin.

На третьем уровне находится «учетная запись» (account), позволяющая пользователям подразделять свои кошельки на отдельные логические учетные записи для ведения бухгалтерского учета или в организационных целях. Например, HD-кошелек может содержать две «учетные записи» (два «счета») для биткойнов: m/44'/0'/0' и m/44'/0'/1'. Каждая учетная запись является корнем отдельного поддерева.

На четвертом уровне change в дереве HD-кошелька создаются две подветви: одна – для создания адресов получателей, другая – для создания адресов приема сдачи. Отметим, что на всех предшествующих уровнях использовалась функция защищенной генерации, а на этом уровне применяется обычная генерация. Это позволяет экспортировать расширяемые открытые ключи на данном уровне дерева для использования в ненадежной среде. Рабочие адреса порождаются HD-кошельком как потомки от четвертого уровня, образующие пятый уровень дерева address_index. Например, третий адрес по-

лучателя для платежей в биткойнах для первой учетной записи должен выглядеть так: M/44'/0'/0'/0/2. В табл. 5.7 приведено еще несколько примеров подобных адресов.

Таблица 5.7. Примеры структуры HD-кошелька по стандарту BIP-44

Путь в HD-кошельке	Описание ключа
M/44'/0'/0'/0/2	Третий открытый ключ для приема платежей в биткойнах для первой учетной записи
M/44'/0'/3'/1/14	Пятнадцатый открытый ключ, соответствующий адресу приема сдачи в биткойнах, для четвертой учетной записи
m/44'/2'/0'/0/1	Второй секретный ключ в основной учетной записи для подписи транзакций, производимых в Litecoin

Глава 6

Транзакции

Введение

Транзакции (transactions) – самая важная часть биткойн-системы. Все прочее в биткойн-системе предназначено для обеспечения создания транзакций, передачи их в сеть, проверки и в конечном итоге для добавления в глобальный реестр транзакций (то есть в структуру данных блокчейна). Транзакции представляют собой структуры данных, в которых закодирована передача ценностей между участниками биткойн-системы. Каждая транзакция – это общедоступная запись в структуре данных блокчейна биткойн-системы, фактически являющейся глобальной бухгалтерской книгой с системой двойной записи.

В этой главе подробно рассматриваются все разнообразные формы транзакций, их содержимое, процедуры их создания и проверки корректности, а также показано, как они становятся частью постоянного реестра всех транзакций. В этой главе термин «кошелек» (wallet) будет использоваться для обозначения программного обеспечения, формирующего транзакции, а не только как база данных ключей.

Транзакции в подробностях

В главе 2 мы рассматривали транзакцию Алисы, используемую для оплаты чашки кофе в кафе Боба, применяя в учебных целях проводник по блокам (рис. 6.1).

Приложение проводника по блокам (block explorer) показывает транзакцию с «адреса» Алисы на «адрес» Боба. Это весьма упрощенное визуальное представление того, что содержится в транзакции. В действительности, как мы увидим в данной главе, бóльшая часть выводимой информации создана проводником по блокам, а не взята из транзакции.

Рис. 6.1 ❖ Транзакция Алисы – оплата в кафе Боба

Транзакции – что внутри

В действительности внутреннее содержимое транзакции значительно отличается от визуального представления транзакции, сформированного типичным проводником по блокам. Фактически большинство конструкций высокого уровня, которые мы наблюдаем в различных версиях пользовательского интерфейса биткойн-приложений, на самом деле не существует в биткойн-системе.

Мы можем воспользоваться интерфейсом командной строки Bitcoin Core (`getrawtransaction` и `decoderawtransaction`) для извлечения транзакции Алисы в настоящем, «сыром» виде, декодировать ее и посмотреть, что она содержит. Результат будет следующим:

```
{
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid":
"7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204
b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d1
72787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
```

```
      "value": 0.01500000,
      "scriptPubKey": "OP_DUP OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value": 0.08450000,
      "scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
    }
  ]
}
```

Возможно, вы обратили внимание на некоторые особенности этой транзакции, главной из которых является отсутствие некоторых элементов. Где адрес Алисы? Где адрес Боба? Где входные данные, то есть число 0.1, «отправленное» Алисой? В биткойн-системе нет монет, нет отправителей, нет получателей, нет балансов, нет учетных записей, нет адресов. Все эти элементы формируются на более высоком уровне для удобства пользователя, чтобы сделать все происходящее более доступным для понимания.

Вероятно, вы также заметили множество непонятных, не поддающихся расшифровке полей и строк в шестнадцатеричном формате. Сейчас не стоит беспокоиться об этом, потому что в данной главе будут подробно описаны все поля, показанные в вышеприведенном примере.

Входные и выходные данные транзакции

Основополагающим элементом конструкции биткойн-транзакции являются выходные данные (transaction output). Выходные данные транзакции – это неделимые фрагменты биткойнов (как денежной единицы), записанные в структуру данных блокчейна и признанные корректными всей сетью. Полноценные узлы биткойн-системы отслеживают все доступные выходные данные, которые могут расходоваться. Эти выходные данные называются неизрасходованными выходными данными транзакций (unspent transaction outputs), сокращенно UTXO. Полный список всех данных UTXO называется UTXO-набором (UTXO set) и в настоящее время насчитывает миллионы записей UTXO. UTXO-набор увеличивается всякий раз, когда создаются новые данные UTXO, и уменьшается по мере расходования UTXO. Каждая транзакция представляет изменение (переход между состояниями) в UTXO-наборе.

Когда мы говорим, что кошелек «получил» биткойны, это означает, что кошелек обнаружил данные UTXO, которые можно расходовать с помощью одного из ключей, управляемых этим кошельком. Таким образом, пользовательский «баланс» биткойнов представляет собой сумму всех данных UTXO, которые кошелек этого пользователя может израсходовать и которые, возможно, будут распределены по сотням транзакций и сотням блоков. Концепция баланса создается приложением кошелька. Кошелек вычисляет баланс пользователя, по-

следовательно просматривая структуру данных блокчейна и суммируя значения всех данных UTXO, которые может расходовать этот кошелек с помощью управляемых им ключей. Большинство кошельков поддерживает базу данных или использует сервис базы данных для хранения оперативной ссылки на набор всех данных UTXO, которые они могут расходовать с помощью своих ключей.

В выходных данных транзакций может содержаться произвольное (целое) значение, обозначенное как множитель (кратное число) для единиц «сатоши» (satoshi). Так же как суммы в долларах и рублях могут содержать две позиции после десятичной точки (запятой) для записи центов (копеек), так и суммы в биткойнах могут включать до восьми позиций после десятичной точки (запятой) для записи сатоши. Выходные данные могут содержать любое произвольное значение, но после создания они становятся неделимыми. Это очень важная характеристика выходных данных, которую нужно особо выделить: выходные данные – это дискретные и неделимые единицы ценности, выраженные в целочисленных сатоши. Неизрасходованные выходные данные могут быть включены в любую другую транзакцию только как единое целое.

Если данные UTXO больше, чем требуемое значение транзакции, то даже в этом случае они непременно должны расходоваться как единое целое, но в этой же транзакции должна быть сгенерирована сдача (change). Другими словами, если вы имеете данные UTXO в размере 20 биткойнов, а нужно заплатить только 1 биткойн, то транзакция все равно должна израсходовать все 20 биткойнов UTXO и сгенерировать два фрагмента выходных данных: один – для оплаты суммы в 1 биткойн указанному получателю, другой – для выплаты 19 биткойнов сдачи, возвращаемой в ваш кошелек. Следствием неделимой природы выходных данных транзакций является тот факт, что большинство биткойн-транзакций будет вынуждено генерировать сдачу.

Представим себе человека, покупающего некий напиток за 1.50 доллара и роющегося в своем кошельке в поисках монет и купюр, составляющих требуемую сумму. Покупатель наверняка постарается набрать сумму без сдачи, если это возможно (например, долларовая купюра и две монеты в 25 центов), или соберет горсть мелких монет (шесть монет по 25 центов), но если необходимо, то выберет более крупную купюру, например в 5 долларов. Если покупатель передает владельцу магазина (или продавцу) бóльшую сумму, скажем, те же 5 долларов, то вполне обоснованно ожидает сдачу в размере 3.50 доллара, которые он(а) вернет в свой кошелек и сделает доступными для последующих транзакций.

Подобным образом должны создаваться и биткойн-транзакции из пользовательских данных UTXO в тех номиналах, которые доступны пользователю. Пользователи не могут делить данные UTXO пополам – так же, как не могут разрезать долларовую купюру пополам и использовать обе части как отдельные денежные единицы. Приложение кошелька обычно выбирает из всех доступных пользовательских данных UTXO те, которые составляют сумму, большую или равную требуемой сумме транзакции.

Как и в реальной жизни, биткойн-приложение может применять несколько разных стратегий для формирования суммы оплаты за покупку: объединение нескольких более мелких элементов, поиск суммы без сдачи или использование одного элемента данных, более крупного, чем сумма транзакции, и формирование сдачи. Все эти сложные манипуляции с доступными данными UTXO кошелек выполняет автоматически и незаметно для пользователей. Это имеет значение только в том случае, если вы сами программно формируете транзакции на низком уровне из данных UTXO.

Транзакции расходуют ранее записанные неизрасходованные выходные данные предыдущих транзакций и создают новые выходные данные, которые могут расходоваться будущими транзакциями. Таким образом, части стоимости биткойнов постоянно перемещаются от владельца к владельцу в цепочке транзакций, потребляющих и создающих данные UTXO.

Исключением из цепочек входных и выходных данных является особый тип транзакций, называемый coinbase-транзакцией. Coinbase-транзакция – это самая первая транзакция в каждом блоке, которая размещается майнером-«победителем» и создает новые биткойны, выплачиваемые ему как вознаграждение за успешный майнинг. Эта особенная coinbase-транзакция не потребляет данных UTXO, вместо этого она принимает специализированный тип входных данных, называемый «coinbase». Именно таким способом создаются новые денежные единицы биткойны в процессе майнинга, но более подробно мы рассмотрим этот процесс в главе 10.

Что было раньше? Входные или выходные данные, курица или яйцо? Строго говоря, выходные данные появляются раньше, так как coinbase-транзакции, которые генерируют новые биткойны, фактически не принимают никаких входных данных и создают выходные данные из ничего.

Выходные данные транзакции

Каждая биткойн-транзакция создает выходные данные, которые записываются в реестр биткойн-системы. Почти все эти выходные данные, за одним исключением (см. раздел «Запись выходных данных (RETURN)» главы 7), создают доступные для расходования части биткойнов, называемые UTXO и признанные как корректные всей сетью и доступные для владельца, который может расходовать их в следующей транзакции.

Данные UTXO отслеживаются каждым полноценным узлом биткойн-клиента в UTXO-наборе. Новые транзакции потребляют (расходуют) один или несколько элементов (записей) выходных данных из UTXO-набора.

Выходные данные транзакции состоят из двух частей:

- количество биткойнов, выраженное в *сатоши* (*satoshi*), то есть в минимальной единице биткойн-системы;
- криптографическая головоломка, определяющая условия, требуемые для расходования этих выходных данных.

Криптографическая головоломка также известна под названием «*блокирующий скрипт*» (*locking script*), «*скрипт-свидетельство*» (*witness script*) или `scriptPubKey`.

Язык скриптов транзакций, используемый в блокирующих скриптах, упомянутых выше, подробно рассматривается в разделе «Скрипты транзакций и язык скриптов» текущей главы.

Теперь вернемся к транзакции Алисы (показанной выше в разделе «Транзакции – что внутри») и попробуем определить в ней выходные данные. В кодировке JSON выходные данные размещаются в массиве (списке) с именем `vout`:

```
"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
OP_EQUALVERIFY
    OP_CHECKSIG"
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
OP_EQUALVERIFY OP_CHECKSIG",
  }
]
```

Здесь можно видеть, что транзакция содержит два фрагмента выходных данных. Каждый фрагмент выходных данных определяется значением и криптографической головоломкой. В кодировке, используемой инструментами Bitcoin Core, значение выводится в биткойнах, но в самой транзакции это значение записано как целое число, выраженное в сатоши. Второй частью каждого фрагмента выходных данных является криптографическая головоломка, которая устанавливает условия для расходования этой суммы. Инструменты Bitcoin Core обозначают эту часть как `scriptPubKey` и показывают легко читаемое представление соответствующего скрипта.

Вопросы блокировки и разблокировки данных UTXO будут обсуждаться несколько позже в текущей главе, в разделе «Формирование структуры скрипта (Lock + Unlock)». Язык, используемый для скриптов в части `scriptPubKey`, рассматривается в разделе «Скрипты транзакций и язык скриптов» текущей главы. Но, прежде чем мы перейдем к этим темам, необходимо понять общую структуру входных и выходных данных транзакций.

Сериализация транзакций – выходные данные

При передаче транзакций по сети или при обмене ими между приложениями выполняется *сериализация* (*serialization*) транзакций. Сериализация – это процесс преобразования внутреннего представления структуры данных в формат, который можно передавать по одному байту за раз. Такой формат передачи называется потоком байтов (byte stream). Наиболее часто сериализация ис-

пользуется для кодирования структур данных при передаче их по сети или при сохранении их в файле. Формат сериализации для выходных данных транзакций показан в табл. 6.1.

Таблица 6.1. Формат сериализации для выходных данных транзакций

Размер	Поле	Описание
8 байтов (порядок от младшего к старшему)	Amount (Сумма)	Сумма биткойнов в сатоши (10^{-8} биткойнов)
1–9 байтов (VarInt)	Locking-Script Size (Размер блокирующего скрипта)	Длина блокирующего скрипта (см. ниже) в байтах
Переменный	Locking-Script (Блокирующий скрипт)	Скрипт, определяющий условия, необходимые для расходования выходных данных

Большинство библиотек и программных инструментальных сред для работы с биткойном не использует для внутреннего хранения потоки байтов, так как при этом требуется выполнение сложного синтаксического разбора каждый раз, когда необходимо получить доступ к какому-то отдельному полю. Для удобства обработки и чтения библиотеки для работы с биткойном организуют внутреннее хранение транзакций в структурах данных (обычно в объектно-ориентированных структурах).

Процесс преобразования из представления транзакции в виде потока байтов во внутреннее представление библиотеки в виде некоторой структуры данных называется *десериализацией* (*deserialization*), или *синтаксическим разбором* (*парсингом*), *транзакции* (*transaction parsing*). Процесс обратного преобразования в поток байтов для передачи по сети, для хэширования или для сохранения в файле на диске называется *сериализацией* (*serialization*). Большинство библиотек для работы с биткойном имеет встроенные функции для сериализации и десериализации транзакций.

В качестве упражнения попробуйте вручную декодировать транзакцию Алисы из сериализованной формы в шестнадцатеричных кодах и найти некоторые элементы, которые мы наблюдали ранее. Чтобы вам было легче разобраться, секция, содержащая два фрагмента выходных данных, выделена в следующем примере 6.1.

Пример 6.1 ❖ Транзакция Алисы – сериализованная и представленная в шестнадцатеричной нотации

```
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73
4d2804fe65fa35779000000008b483045022100884d142d86652a3f47
ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039
ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813
01410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc1
7b4a10fa336a8d752adfffffffff02 60e3160000000001976a914ab6
8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef800000000000
1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac 00000000
```

Несколько советов:

- в выделенной секции содержатся два фрагмента выходных данных, каждый из которых сериализован в соответствии с табл. 6.1;
- значение 0.015 биткойна равно 1 500 000 сатоши. В шестнадцатеричном формате это число `16 e3 60`;
- в сериализованной транзакции число `16 e3 60` закодировано с порядком байтов от младшего к старшему (little-endian), поэтому выглядит как `60 e3 16`;
- длина скрипта `scriptPubKey` равна 25 байтам, в шестнадцатеричном формате это число `19`.

Входные данные транзакции

Входные данные транзакции определяют (по ссылке), какие данные UTXO будут расходоваться, а также предоставляют доказательство права владения с помощью разблокирующего скрипта.

Для создания транзакции кошелек выбирает из всех выходных данных, которыми он управляет, данные UTXO со значением, достаточным для требуемого платежа. Иногда достаточно одного элемента данных UTXO, иногда нужно взять несколько таких элементов. Для каждого элемента данных UTXO, расходуемого при формировании конкретного платежа, кошелек создает один элемент входных данных, указывающий на выбранные данные UTXO, и разблокирует его с помощью соответствующего скрипта.

Рассмотрим все компоненты входных данных более подробно. Первая часть входных данных – указатель на выходные данные UTXO со ссылкой на хэш транзакции и номер последовательности, в которой выходные данные UTXO записаны в структуре данных блокчейна. Вторая часть – разблокирующий скрипт, который кошелек формирует для соответствия набору условий расходования, заданных в выходных данных UTXO. Чаще всего разблокирующий скрипт представляет собой цифровую подпись и открытый ключ, подтверждающий право владения биткойнами. Тем не менее не все разблокирующие скрипты содержат цифровые подписи. Третья часть – номер последовательности, который будет объяснен позже.

Рассмотрим пример из раздела «Транзакции – что внутри» в начале текущей главы. Входные данные транзакции показаны в виде массива (списка) с именем `vin`:

```
"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204
b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d1
72787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
```

```
    "sequence": 4294967295
  }
]
```

Здесь можно видеть, что в списке содержится только один элемент входных данных (поскольку один элемент выходных данных UTXO содержит значение, достаточное для выполнения рассматриваемого платежа). Этот элемент содержит четыре компонента:

- идентификатор (ID) транзакции, ссылающийся на транзакцию, содержащую расходуемые данные UTXO;
- индекс выходных данных (vout), определяющий, на какой элемент данных UTXO из этой транзакции указывает ссылка (первому элементу соответствует нулевой индекс);
- скрипт `scriptSig`, выполняющий условия, установленные в элементе данных UTXO, для разблокировки и расходования этого элемента;
- номер последовательности (будет объяснен позже).

В транзакции Алисы входные данные ссылаются на идентификатор транзакции:

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18
```

и на индекс выходных данных `0` (то есть на первый элемент данных UTXO, созданный этой транзакцией). Кошелек Алисы создает разблокирующий скрипт, в первую очередь извлекая указанные данные UTXO, проверяя соответствующий блокирующий скрипт, затем используя эти компоненты для формирования разблокирующего скрипта, необходимого для соответствия заданным условиям.

Изучая входные данные, вы, вероятно, заметили, что мы ничего не знаем об указанных данных UTXO, кроме ссылки на транзакцию, содержащую эти данные. Нам не известно значение транзакции (сумма в сатоши), не известен блокирующий скрипт, устанавливающий условия расходования этой суммы. Для получения требуемой информации необходимо обратиться к указанным данным UTXO и извлечь содержащуюся в них транзакцию. Отметим, что поскольку значение входных данных не задано явно, необходимо также использовать указанные данные UTXO для вычисления платежей, которые будут включены в создаваемую транзакцию (см. раздел «Оплата транзакций» ниже в текущей главе).

Но не только кошельку Алисы потребуется извлечение данных UTXO, на которые ссылаются входные данные. После опубликования транзакции в сети каждый проверяющий узел также должен будет извлечь данные UTXO, на которые ссылается эта транзакция, чтобы проверить ее корректность.

Транзакции сами по себе в отрыве от других транзакций выглядят неполными, потому что отсутствует контекст. Транзакции ссылаются на данные UTXO в своих входных данных, но без извлечения этих данных UTXO мы не сможем узнать значение входных данных или заданные условия блокирования. При написании программного обеспечения для биткойна каждый раз, когда вы де-

кодируете транзакцию, чтобы проверить ее, или определить отчисления, или проверить разблокирующий скрипт, сначала необходимо извлечь указанные данные UTXO из структуры данных блокчейна, чтобы сформировать контекст, подразумеваемый, но реально не присутствующий в ссылках на UTXO из входных данных. Например, для вычисления суммы оплаты транзакции необходимо знать сумму значений входных и выходных данных. Но это значение невозможно определить без извлечения данных UTXO, на которые ссылаются входные данные. Поэтому операция, кажущаяся простой, такая как определение отчислений в одной транзакции, в действительности требует выполнения нескольких шагов и получения данных из нескольких транзакций.

Можно воспользоваться той же последовательностью команд Bitcoin Core, которую мы применяли для получения транзакции Алисы (`getrawtransaction` и `decoderawtransaction`). С их помощью мы получаем данные UTXO, на которые ссылаются рассматриваемые выше входные данные:

```
"vout": [
   {
     "value": 0.10000000,
     "scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG"
   }
 ]
```

Здесь мы видим, что данные UTXO состоят из значения 0.1 BTC и блокирующего скрипта (`scriptPubKey`), который содержит строку `"OP_DUP OP_HASH160..."`.

Для полного понимания транзакции Алисы необходимо извлечь все предыдущие транзакции, на которые ссылаются входные данные. Функция, извлекающая предыдущие транзакции и неизрасходованные выходные данные транзакций, применяется весьма часто, поэтому имеется практически в каждой библиотеке, поддерживающей работу с биткойном, и в каждом прикладном программном интерфейсе (API).

Сериализация транзакций – входные данные

При сериализации транзакций для передачи их по сети выполняется преобразование входных данных в поток байтов, как показано в табл. 6.2.

Таблица 6.2. Сериализация входных данных транзакции

Размер	Поле	Описание
32 байта	Хэш транзакции	Указатель на транзакцию, содержащую расходуемые данные UTXO
4 байта	Индекс выходных данных	Номер индекса расходуемых данных UTXO. Первому элементу соответствует индекс 0
1–9 байтов (VarInt)	Размер разблокирующего скрипта	Длина в байтах разблокирующего скрипта (см. ниже)
Переменный	Разблокирующий скрипт	Скрипт, который выполняет условия, заданные в блокирующем скрипте данных UTXO
4 байта	Номер последовательности	Используется для синхронизации блокировки или ее запрещения (0xFFFFFFFF)

Так же, как и для выходных данных, проверим, сможем ли мы найти входные данные из транзакции Алисы в сериализованном формате. Сначала декодируем входные данные:

```
"vin": [
  {
    "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
    "vout": 0,
    "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204
b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d1
72787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
    "sequence": 4294967295
  }
],
```

Теперь попытайтесь определить эти поля в сериализованной шестнадцатеричной кодировке, приведенной в примере 6.2.

Пример 6.2 ❖ Транзакция Алисы, сериализованная и представленная в шестнадцатеричной нотации

```
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73
4d2804fe65fa35779000000008b483045022100884d142d86652a3f47
ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039
ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813
01410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade84
16ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc1
7b4a10fa336a8d752adfffffffff0260e31600000000001976a914ab6
8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef800000000000
1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000 000
```

Подсказки:

- идентификатор (ID) транзакции сериализован с обратным порядком байтов, поэтому он начинается с (шестнадцатеричного) числа 18 и заканчивается числом 79;
- индекс выходных данных – это 4-байтовая группа нулей, ее легко обнаружить;
- длина скрипта `scriptSig` равна 139 байтам, или 8b в шестнадцатеричном формате;
- для номера последовательности установлено значение FFFFFFFF, это число тоже легко найти.

Оплата транзакций

В большинство транзакций включены суммы оплаты (отчислений) за выполнение транзакции, служащие компенсацией майнерам биткойнов за поддержку безопасности биткойн-сети. Сами по себе эти отчисления служат в качестве механизма защиты, делая экономически невыгодным для атакующих запол-

нение сети огромным количеством транзакций. Более подробно процесс майнинга, а также отчисления (оплата транзакций) и вознаграждения, принимаемые майнерами, рассматриваются в главе 10.

В этом разделе рассматривается, как оплата транзакций (сбор за транзакции) включается в обычную транзакцию. Большинство кошельков автоматически вычисляет и добавляет оплату транзакций. Но если вы формируете транзакцию программно или с использованием интерфейса командной строки, то обязательно должны учесть и вручную включить все необходимые оплаты за транзакции.

Отчисления за транзакции служат материальным стимулом для включения (при майнинге) транзакций в очередной блок, но в то же время защитой от злоупотреблений в системе благодаря весьма небольшой сумме оплаты за каждую транзакцию. Оплату за транзакции принимает майнер, сформировавший блок, который осуществляет запись транзакций в структуру данных блокчейна.

Оплата транзакций вычисляется на основе размера транзакции в килобайтах, а не по сумме транзакции в биткойнах. В целом суммы оплаты транзакций устанавливаются на основе рыночных сил (сил спроса/предложения) в биткойн-сети. Майнеры устанавливают приоритеты транзакций с учетом множества различных критериев, в том числе и отчислений (сборов), и могут даже обрабатывать транзакции бесплатно при определенных обстоятельствах. Оплата транзакций влияет на приоритет обработки, то есть транзакция с достаточной суммой оплаты, вероятнее всего, будет включена в очередной блок майнинга, тогда как транзакция с недостаточной суммой оплаты или вообще без оплаты может быть задержана, обработана после пропуска нескольких блоков с целью минимизации накладных расходов или не обработана вообще. Оплата транзакций не обязательна, и транзакции без оплаты, в конце концов, будут когда-нибудь обработаны, но включение в транзакцию достаточной суммы оплаты повышает ее приоритет при обработке.

Способ вычисления суммы оплаты транзакций и учета ее воздействия на систему приоритетов транзакций со временем усовершенствуется. Сначала суммы оплаты транзакций были постоянными для всей сети в целом. Постепенно структура отчислений становилась более свободной, и на нее все большее влияние стали оказывать рыночные силы с учетом мощности сети и объема транзакций. Приблизительно с начала 2016 года ограничения денежной массы биткойнов создали конкуренцию между транзакциями, в результате суммы оплаты транзакций повысились, а быстрая обработка транзакций без оплаты осталась в прошлом. Транзакции с нулевой или чрезвычайно малой суммой оплаты редко включаются в процесс майнинга, а иногда даже не распространяются по сети.

В Bitcoin Core стратегии продвижения транзакций в сети на основе суммы их оплаты устанавливаются параметром `minrelaytxfee`. Текущее значение по умолчанию для этого параметра равно 0.00001 биткойна, или одна сотая милибиткойна за килобайт. Таким образом, по умолчанию транзакции, в кото-

рых указана сумма оплаты менее 0.0001 биткойна, считаются бесплатными и продвигаются в сети только при наличии свободного места в пуле памяти (mempool), в противном случае такие транзакции отбрасываются. Биткойн-узлы могут переписывать стратегию продвижения транзакций в сети по сумме оплаты, изменяя значение параметра `minrelaytxfee`.

Каждый сервис, обеспечивающий работу с биткойнами, который создает транзакции, включая кошельки, обменные площадки, приложения для розничной торговли и т. п., обязательно должен поддерживать реализацию динамической оплаты транзакций. Динамическая оплата транзакций (dynamic fee) может быть реализована сторонним сервисом оценки и вычисления оплаты транзакций или с помощью встроенного алгоритма оценки и вычисления оплаты транзакций. Если вы не уверены в своих силах, начните с использования стороннего сервиса, а по мере накопления опыта можно будет спроектировать и реализовать собственный алгоритм, если потребуется устранение зависимости от третьих сторон.

Алгоритмы оценки оплаты транзакций вычисляют соответствующую сумму на основе общего объема и сумм оплат, предлагаемых «конкурирующими» транзакциями. Диапазон таких алгоритмов достаточно широк: от простейших (вычисление среднего арифметического всех сумм или определение медианной суммы оплаты в последнем блоке) до весьма изощренных (статистический анализ). Алгоритмы производят оценку необходимой суммы (в сатоши за байт), которая обеспечит транзакции высокую вероятность выбора и включения в определенное количество блоков. Большинство сервисов предоставляет пользователям возможность выбора суммы оплаты с высоким, средним или низким приоритетом. Высокий приоритет означает, что пользователь платит бóльшую сумму, но транзакция, вероятнее всего, будет включена в следующий блок. При выборе среднего и низкого приоритетов пользователь платит меньшую сумму, но время подтверждения транзакций увеличивается.

Многие приложения кошельков используют сторонние сервисы для вычисления оплаты транзакций. Одним из наиболее известных подобных сервисов является http://bitcoinfees.21.co, предоставляющий прикладной программный интерфейс и визуальную диаграмму, на которой наглядно показаны суммы оплаты в сатоши/байт для различных приоритетов.

Сейчас статическая оплата транзакций уже нежизнеспособна в биткойн-сети. Кошельки, устанавливающие статическую оплату транзакций, разочаруют пользователей, потому что транзакции будут часто «застревать» и оставаться неподтвержденными. Пользователи, не понимающие механизма биткойн-транзакций и их оплаты, будут встревожены этими «застреваниями», поскольку им покажется, что деньги потеряны безвозвратно.

Диаграмма на рис. 6.2 показывает оценку в реальном времени оплат транзакций с шагом в 10 сатоши/байт и предполагаемое время подтверждения (в минутах и в количестве блоков) для транзакций с суммами оплат в каждом диапазоне. Для каждого диапазона сумм оплат (например, 61–70 сатоши/байт) две

горизонтальные полосы показывают количество неподтвержденных транзакций (1405) и общее количество транзакций за последние 24 часа (102 975), суммы оплаты которых находятся в этом диапазоне. По этой диаграмме на тот момент времени рекомендуемая оплата с высоким приоритетом была равна 80 сатоши/байт, то есть транзакция с такой суммой оплаты с большой вероятностью будет включена в ближайший очередной блок (с нулевой задержкой ожидания блока). Медианный (срединное значение) размер транзакций равен 226 байтам, следовательно, рекомендуемая оплата транзакции такого размера должна составлять 18 080 сатоши (0.00018080 BTC).

Оценочные данные оплаты транзакций можно получить с помощью простого программного интерфейса HTTP REST API по адресу https://bitcoinfees.21.co/api/v1/fees/recommended. Например, в командной строке с помощью утилиты `curl`:

```
$ curl https://bitcoinfees.21.co/api/v1/fees/recommended

{"fastestFee":80,"halfHourFee":80,"hourFee":60}
```

В этом примере интерфейс HTTP REST API возвращает JSON-объект с текущими оценками сумм оплаты транзакций для самого быстрого подтверждения (`fastestFee`), для подтверждения в ближайших трех блоках (`halfHourFee`) или шести блоках (`hourFee`) в сатоши/байт.

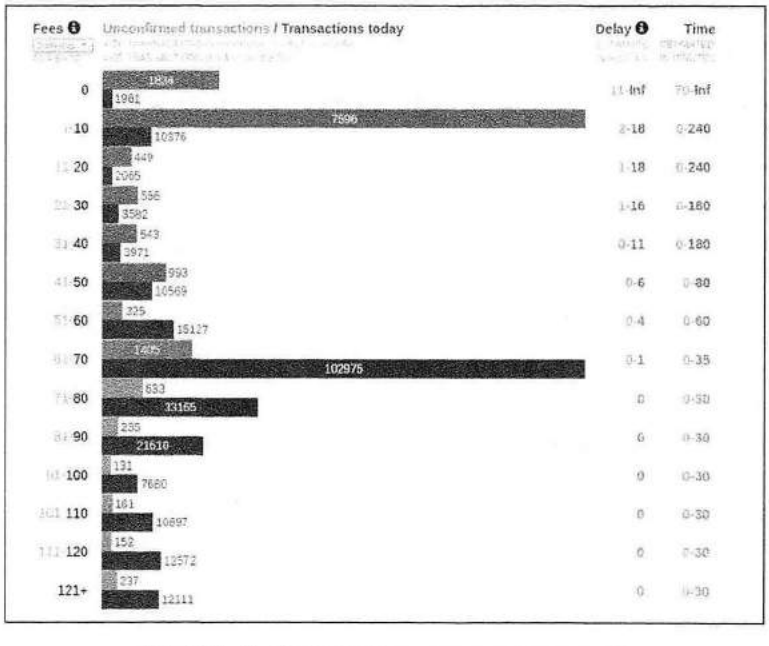


Рис. 6.2 ❖ Оценка сумм оплаты транзакций с помощью сервиса bitcoinfees.21.co

Добавление сумм оплаты в транзакции

Структура данных транзакции не содержит специального поля для оплаты. Вместо этого предполагается, что сумма оплаты вычисляется как разность между суммой входных данных и суммой выходных данных. Остаток после вычитания всех выходных данных из всех входных данных представляет собой оплату транзакции, передаваемую майнерам:

```
Fees = Sum(Inputs) - Sum(Outputs)
```

Этот элемент транзакции определен весьма нечетко, но очень важно его полное понимание, потому что если вы формируете собственную транзакцию, то непременно должны быть уверены в том, что по неосторожности не включили слишком большую сумму оплаты, не израсходовав всех входных данных. Это означает, что следует учесть (создать учетные записи) все входные данные, если необходимо, создать учетную запись для сдачи, иначе вы отдадите майнерам слишком большие «чаевые».

Например, если вы расходуете данные UTXO размером в 20 биткойнов для формирования платежа с суммой в 1 биткойн, то обязательно должны предусмотреть возврат 19 биткойнов сдачи в собственный кошелек. В противном случае «оставшиеся» 19 биткойнов будут считаться оплатой транзакции и передаются майнеру, который включает вашу транзакцию в блок. Конечно, вы получите высочайший приоритет при обработке и сделаете майнера очень счастливым, но, вероятнее всего, это совсем не то, что вы хотели сделать в действительности.

Если вы забыли добавить выходные данные для возврата сдачи в транзакцию, формируемую вручную, то вся сумма сдачи уходит на оплату транзакции. «Сдачи не надо» – возможно, это совсем не то, что вы хотели.

Рассмотрим, как это работает на практике, и снова обратимся к примеру оплаты чашки кофе Алисой. Алиса хочет потратить 0.015 биткойна, чтобы заплатить за кофе. Для быстрой обработки транзакции она хочет включить оплату, скажем, в размере 0.001. При этом общая сумма транзакции составит 0.016. Следовательно, кошелек Алисы должен найти набор данных UTXO, составляющих в сумме 0.016 биткойна или больше, и при необходимости определить размер сдачи. Допустим, кошелек обнаружил доступные данные UTXO с 0.2 биткойна. Необходимо израсходовать эти данные UTXO, создавая один фрагмент выходных данных для выплаты кафе Боба 0.015, второй фрагмент выходных данных для возврата 0.184 биткойна как сдачи в кошелек Алисы и оставляя 0.001 биткойна нераспределенными, то есть как предполагаемую сумму оплаты этой транзакции.

Рассмотрим другой пример. Эухения, директор благотворительного фонда помощи детям на Филиппинах, завершила кампанию по сбору средств для приобретения школьных учебников. Она получила несколько тысяч пожертвований небольших сумм от людей со всего мира. Общая сумма составила 50 бит-

койнов, и кошелек Эухении заполнен очень малыми платежами (UTXO). Далее она намерена приобрести несколько сотен школьных учебников у местного издателя, принимающего оплату в биткойнах.

Приложение кошелька Эухении пытается сформировать единственную транзакцию с более крупной суммой платежа, составленной исключительно из набора доступных данных UTXO, который представляет собой множество более мелких сумм. Это означает, что итоговая транзакция будет сформирована из сотни с лишним небольших значений данных UTXO в качестве входных данных и только одного фрагмента выходных данных как оплата заказа книг в издательстве. Транзакция с таким большим количеством фрагментов входных данных будет иметь размер более одного килобайта, возможно, даже несколько килобайтов. Поэтому она потребует большей суммы оплаты, чем транзакция «медианного» (срединного) размера.

Приложение кошелька Эухении вычислит соответствующую сумму оплаты, оценив общий размер транзакции и умножив его на выбранную плату за килобайт. Многие кошельки немного переплачивают за крупные транзакции, чтобы обеспечить их обработку как можно быстрее. Более высокая оплата – это не следствие того, что Эухения тратит крупную сумму денег, это происходит потому, что ее транзакция более сложна и имеет большой размер – оплата не зависит от суммы транзакции в биткойнах.

Скрипты транзакций и язык Script

Язык скриптов биткойн-транзакций под названием Script – это Forth-подобный язык с обратной польской нотацией и с механизмом выполнения, основанным на стеке. Если такое описание вам совершенно непонятно, значит, вы не изучали языки программирования в 1960-х гг., но это не имеет значения – в текущей главе все будет объяснено в деталях. На этом языке написаны и блокирующие скрипты, расположенные в данных UTXO, и разблокирующие скрипты. При проверке корректности транзакции разблокирующий скрипт в каждом фрагменте входных данных выполняется вместе с соответствующим блокирующим скриптом, чтобы убедиться в соблюдении условия расходования средств.

Script – очень простой язык, специализированный для конкретной области применения и выполняемый на многих типах аппаратных средств, даже на самых простых, таких как встроенные устройства. Он требует минимальной обработки и не способен на большинство хитроумных действий, которые могут выполнять современные языки программирования. При его использовании для проверки операций с деньгами, создаваемыми программным путем, такой подход обеспечивает функцию защиты.

Сегодня большинство транзакций, обрабатываемых в биткойн-сети, имеет форму «Платеж на биткойн-адрес Боба» и основано на скрипте, называемом Pay-to-Public-Key-Hash (P2PKH). Но биткойн-транзакции не ограничены именно этим типом скрипта. В действительности блокирующие скрипты могут быть написаны для создания огромного разнообразия сложных составных условий.

Чтобы уверенно разбираться в таких более сложных скриптах, сначала необходимо понять основы скриптов транзакций и языка скриптов.

В этом разделе будут рассматриваться основные компоненты языка скриптов биткойн-транзакций, а также показано, как их можно использовать для формирования простых условий расходования средств и каким образом эти условия могут быть выполнены разблокирующими скриптами.

Проверка корректности биткойн-транзакций основана не на каком-либо конкретном статическом шаблоне, вместо этого она выполняется средствами языка скриптов. Этот язык предоставляет возможность формирования практически неограниченного количества разнообразных условий. Благодаря языку скриптов были созданы столь мощные возможности «программируемых денег».

Неполнота по Тьюрингу

Язык скриптов для биткойн-транзакций содержит множество операторов, но преднамеренно ограничен в очень важном аспекте – в нем нет циклов и прочих сложных средств управления потоком выполнения, кроме оператора управления потоком выполнения по условию. Это означает, что язык не является Тьюринг-полным, то есть сложность скриптов ограничена, а время их выполнения предсказуемо. Script не является языком программирования общего назначения. Преднамеренно созданные ограничения гарантируют, что этот язык нельзя использовать для создания бесконечного цикла или какой-либо другой формы «логической бомбы», которую можно спрятать в транзакции, с тем чтобы организовать атаку типа DoS (Denial-of-Service) против биткойн-сети. Напомним, что корректность любой транзакции проверяется каждым полноценным узлом биткойн-сети. Ограниченность средств языка защищает механизм проверки транзакций и практически устраняет его уязвимость.

Верификация без сохранения состояния

Язык скриптов для биткойн-транзакций не сохраняет состояния (stateless), то есть в нем нет понятия состояния до выполнения скрипта или состояния, сохраненного после выполнения скрипта. Таким образом, вся информация, необходимая для выполнения конкретного скрипта, содержится в самом этом скрипте. Любой скрипт выполняется предсказуемым образом на любой системе. Если ваша система подтверждает корректность скрипта, то вы можете быть уверенным в том, что все прочие системы в биткойн-сети также подтвердят его корректность, то есть проверенная корректная транзакция корректна для всех и это известно всем. Такая предсказуемость результатов является важнейшим преимуществом биткойн-системы.

Формирование структуры скрипта (Lock + Unlock)

Механизм проверки корректности биткойн-транзакций зависит от двух типов скриптов проверки транзакций: блокирующего скрипта (locking script) и разблокирующего скрипта (unlocking script).

Блокирующий скрипт – это условие расходования средств, размещенное в выходных данных: он определяет условия, которые непременно должны быть выполнены для дальнейшего расходования соответствующего фрагмента выходных данных. Изначально блокирующий скрипт получил название *scriptPubKey*, потому что обычно содержал открытый ключ или биткойн-адрес (хэш открытого ключа). В этой книге термин «блокирующий скрипт» применяется для обозначения более широкого диапазона возможностей данной скриптовой технологии. В большинстве биткойн-приложений то, что мы называем блокирующим скриптом, в исходном коде выглядит как `scriptPubKey`. Кроме того, вы встретите упоминание о блокирующем скрипте в качестве *скрипта-доказательства* (*witness script*) (см. приложение Г) или в более общем смысле как обозначение *криптографической головоломки* (*cryptographic puzzle*). Все эти термины обозначают один и тот же объект на разных уровнях абстракции.

Разблокирующий скрипт – это скрипт, который «решает» или выполняет условия, заданные блокирующим скриптом, размещенным в выходных данных, и обеспечивает возможность расходования этих выходных данных. Разблокирующие скрипты являются обязательной частью входных данных каждой транзакции. В большинстве случаев они содержат цифровую подпись, созданную кошельком пользователя по его секретному ключу. Изначально разблокирующий скрипт получил название *scriptSig*, так как обычно содержал цифровую подпись (digital signature). В большинстве биткойн-приложений разблокирующий скрипт в исходном коде определяется как scriptSig. Кроме того, вы встретите упоминание о разблокирующем скрипте в качестве *свидетельства* (*доказательства*) (*witness*) (см. приложение Г). В этой книге термин «разблокирующий скрипт» применяется для обозначения более широкого диапазона скриптов, выполняющих требования блокирующих скриптов, поскольку не все разблокирующие скрипты обязаны содержать цифровые подписи.

Каждый узел проверки биткойнов будет проверять корректность транзакций, выполняя блокирующие и разблокирующие скрипты совместно. Каждый фрагмент входных данных содержит разблокирующий скрипт и ссылается на ранее существующие данные UTXO. Программное обеспечение, отвечающее за проверку корректности, копирует разблокирующий скрипт, извлекает данные UTXO, на которые ссылаются входные данные, затем копирует блокирующий скрипт из полученных данных UTXO. Затем блокирующий и разблокирующий скрипты выполняются последовательно. Входные данные считаются корректными, если разблокирующий скрипт выполняет условия блокирующего скрипта (см. раздел «Раздельное выполнение разблокирующего и блокирующего скриптов» ниже в текущей главе). Все фрагменты входных данных проверяются независимо друг от друга, как часть общей проверки корректности всей транзакции в целом.

Отметим, что данные UTXO записаны и постоянно хранятся в структуре данных блокчейна, следовательно, они неизменяемы и защищены от некорректных (обманных) попыток расходования с помощью ссылки на них в новой транзакции. Только корректная транзакция, которая правильно выполняет ус-

ловия, заданные в выходных данных, приводит к тому, что эти выходные данные считаются «израсходованными» и удаляются из набора неизрасходованных выходных данных транзакций (UTXO set).

На рис. 6.3 показан пример разблокирующего и блокирующего скриптов для самого общего типа биткойн-транзакции (платеж на хэш открытого ключа). Здесь можно видеть комбинированный скрипт, полученный в результате объединения разблокирующего и блокирующего скриптов перед выполнением проверки.

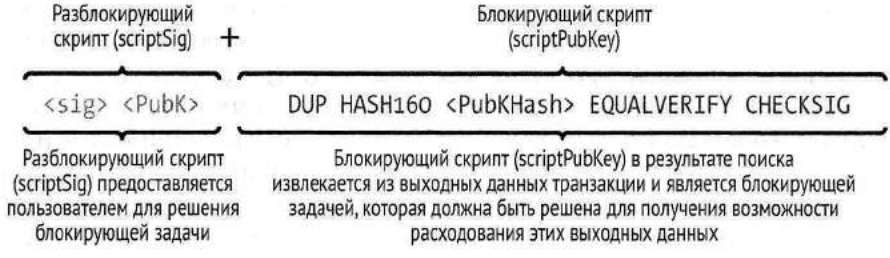


Рис. 6.3 ❖ Объединение scriptSig и scriptPubKey для выполнения проверочного скрипта транзакции

Стек выполнения скрипта

Скриптовый язык биткойна называется стековым языком (или языком с механизмом выполнения, основанным на стеке), потому что он использует структуру данных, называемую стеком (stack). Стек представляет собой очень простую структуру данных, которую можно наглядно представить в виде колоды карт. В стеке разрешены две операции: запись в стек (push) и извлечение из стека (pop). Операция записи в стек добавляет элемент на вершину стека. Операция извлечения удаляет самый верхний элемент из стека. Все операции в стеке могут производиться только над самым верхним элементом стека. Эта структура данных также обозначается как очередь типа «Последним пришел, первым вышел» (Last-In-First-Out, LIFO), или LIFO-очередь.

Внутренний механизм языка выполняет скрипт, обрабатывая каждый элемент в порядке слева направо. Числа (данные-константы) записываются в стек. Операторы записи или извлечения с одним или несколькими параметрами из стека выполняют некоторые действия и могут записать результат обратно в стек. Например, оператор `OP_ADD` извлекает два элемента из стека, складывает их и записывает полученную сумму обратно в стек.

Условные операторы вычисляют условие, результатом которого является логическое (boolean) значение `TRUE` (истина) или `FALSE` (ложь). Например, оператор `OP_EQUAL` извлекает два элемента из стека и записывает в стек значение `TRUE` (представленное числом 1), если элементы равны, или значение `FALSE` (представленное числом 0), если элементы не равны. Скрипты биткойн-транзакций, как правило, содержат условный оператор, так что они могут выдавать результат `TRUE`, который означает, что транзакция корректна.

Простой скрипт

Теперь применим на практике все, что мы узнали о скриптах и стеках, и рассмотрим несколько простых примеров.

На рис. 6.4 скрипт `2 3 OP_ADD 5 OP_EQUAL` демонстрирует применение оператора арифметического сложения `OP_ADD`, суммирующего два числа и помещающего результат в стек. Затем следует условный оператор `OP_EQUAL`, который проверяет вычисленную ранее сумму на равенство числу 5. Для краткости префикс `OP_` не показан на пошаговой схеме примера. Более подробно все доступные операторы и функции языка скриптов описаны в приложении Б.

Несмотря на то что большинство блокирующих скриптов ссылается на хэш открытого ключа (по сути, на биткойн-адрес), следовательно, требуются доказательства права владения для расходования денежных средств, подобный скрипт не обязательно должен быть слишком сложным. Любое сочетание блокирующего и разблокирующего скриптов, которое дает в результате значение TRUE, является корректным. Простые арифметические действия, показанные в вышеприведенном примере, также являются вполне допустимым блокирующим скриптом, который можно использовать для блокировки выходных данных транзакции.

Воспользуемся частью этого арифметического примера как блокирующим скриптом:

```
3 OP_ADD 5 OP_EQUAL
```

условия которого могут быть выполнены транзакцией, содержащей входные данные со следующим разблокирующим скриптом:

```
2
```

Программное обеспечение, проверяющее корректность, объединяет разблокирующий и блокирующий скрипты, получая в результате такой скрипт:

```
2 3 OP_ADD 5 OP_EQUAL
```

В пошаговой схеме примера на рис. 6.4 можно видеть, что при выполнении этого скрипта получен результат OP_TRUE, то есть транзакция корректна. Это не просто подтверждение корректности транзакции с использованием скрипта блокировки выходных данных, это означает, что соответствующие данные UTXO могут расходоваться любым пользователем, арифметические способности которого позволяют понять, что число 2 выполняет условие скрипта.

Транзакции корректны, если на вершине стека расположено итоговое значение `TRUE` (записанное в виде `{0x01}`), любое другое ненулевое значение или если стек пуст после выполнения скрипта. Транзакции считаются некорректными, если на вершине стека обнаружено значение `FALSE` (пустое значение нулевой длины, обозначенное как `{}`) или если выполнение скрипта явно прервано оператором, таким как `OP_VERIFY`, `OP_RETURN`, или обнаружен признак конца условного оператора `OP_ENDIF`. Более подробно об этом см. приложение Б.

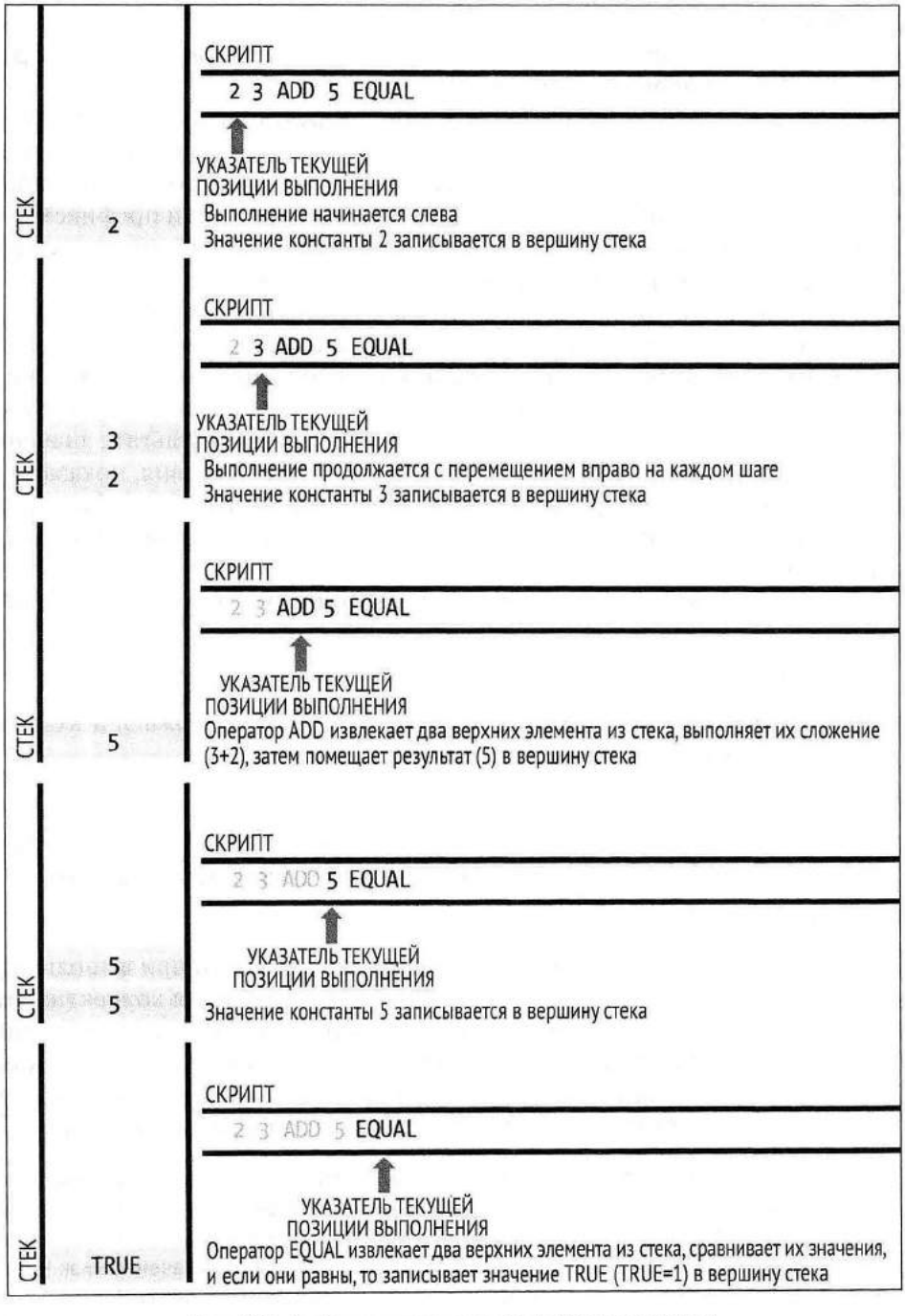


Рис. 6.4 ❖ Скрипт проверки биткойн-транзакции, вычисляющий простое арифметическое выражение

Ниже приведен немного более сложный скрипт, вычисляющий выражение 2 + 7 - 3 + 1. Отметим, что если скрипт содержит несколько операторов в строке, то механизм стека позволяет использовать результат предыдущего оператора для вычисления с помощью следующего оператора:

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Попробуйте проследить ход выполнения этого скрипта с помощью карандаша и бумаги. После завершения выполнения у вас в «стеке» должно остаться значение TRUE.

Раздельное выполнение разблокирующего и блокирующего скриптов

В исходном биткойн-клиенте разблокирующий и блокирующий скрипты объединялись и выполнялись последовательно. По соображениям безопасности этот порядок был изменен в 2010 году из-за уязвимости, позволявшей «неправильному» разблокирующему скрипту записывать данные в стек и нарушать работу блокирующего скрипта. В текущей реализации скрипты выполняются раздельно со стеком, передаваемым между двумя контекстами выполнения, как описано ниже.

Первым выполняется разблокирующий скрипт с использованием стекового механизма выполнения. Если разблокирующий скрипт выполнен без ошибок (то есть в нем нет «висячих» операторов и т. п.), то основной стек (недублированный общий стек) копируется и выполняется блокирующий скрипт. Если результат выполнения блокирующего скрипта со стеком данных, скопированным из разблокирующего скрипта, равен TRUE, то разблокирующий скрипт успешно выполнил условия, заданные блокирующим скриптом, следовательно, входные данные представляют корректную авторизацию для расходования данных UTXO. Любой другой результат выполнения блокирующего скрипта, отличающийся от TRUE, означает, что входные данные некорректны, потому что не выполнили условий, определенных в данных UTXO.

Скрипт Pay-to-Public-Key-Hash (P2PKH)

Подавляющее большинство транзакций, обрабатываемых в биткойн-сети, расходует выходные данные, блокируемые скриптом Pay-to-Public-Key-Hash, или P2PKH. Эти выходные данные содержат блокирующий скрипт с указанием хэша открытого ключа, чаще называемого биткойн-адресом. Выходные данные, заблокированные скриптом P2PKH, можно разблокировать (израсходовать), представив открытый ключ и цифровую подпись, созданную по соответствующему секретному ключу (см. раздел «Цифровые подписи (ECDSA)» ниже в текущей главе).

В качестве примера снова рассмотрим платеж Алисы для кафе Боба. Алиса выполнила оплату в размере 0.015 биткойна на биткойн-адрес кафе. Выходные данные этой транзакции должны содержать блокирующий скрипт следующей формы:

```
OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Фраза `Cafe Public Key Hash` означает биткойн-адрес кафе без кодирования в формате Base58Check. Большинство приложений показало бы *хэш открытого*

ключа (*public key hash*) в шестнадцатеричном формате, а не более привычный биткойн-адрес в формате Base58Check, начинающийся с 1.

Условия приведенного выше блокирующего скрипта могут быть выполнены разблокирующим скриптом, имеющим форму:

```
<Cafe Signature> <Cafe Public Key>
```

Вместе эти два скрипта должны образовать следующий объединенный скрипт проверки корректности:

```
<Cafe Signature> <Cafe Public Key> OP_DUP OP_HASH160
<Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

При выполнении этот объединенный скрипт даст результат TRUE, если и только если разблокирующий скрипт выполнит условия, установленные блокирующим скриптом. Другими словами, результат TRUE будет получен, если в разблокирующем скрипте содержится корректная подпись, сгенерированная из секретного ключа кафе, и эта подпись соответствует хэшу открытого ключа, установленному как препятствие.

На рис. 6.5 и 6.6 (в двух частях) показана схема пошагового выполнения объединенного скрипта, доказывающего корректность рассматриваемой транзакции.

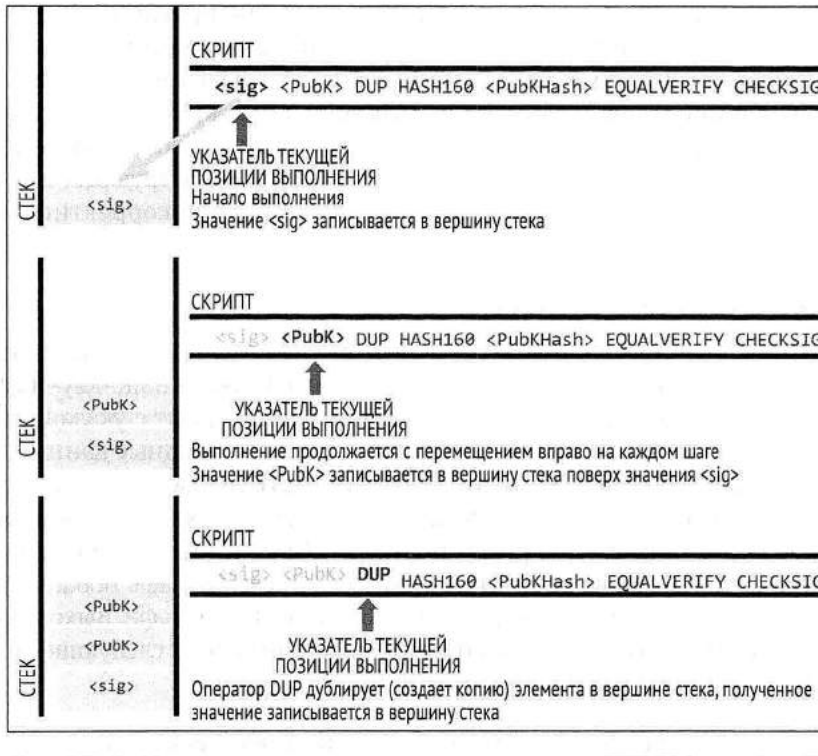


Рис. 6.5 ❖ Выполнение скрипта для транзакции типа P2PKH (часть 1 из 2)

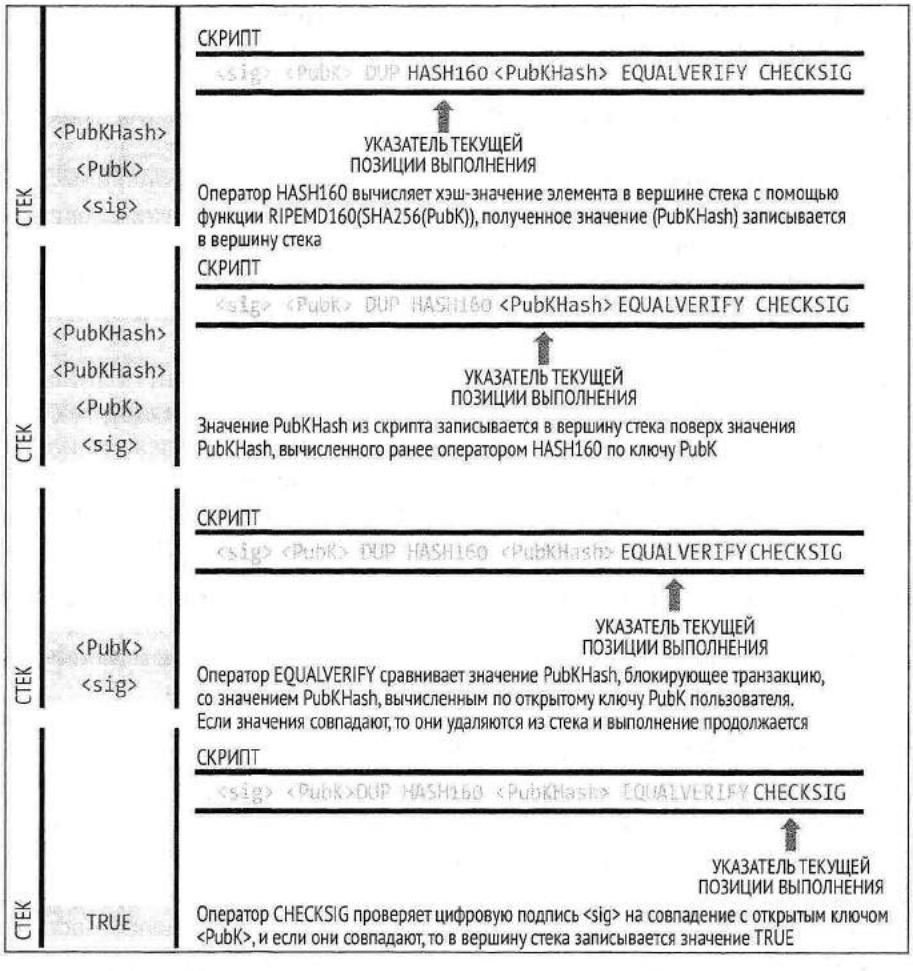


Рис. 6.6 ❖ Выполнение скрипта для транзакции типа P2PKH (часть 2 из 2)

Цифровые подписи (ECDSA)

До настоящего момента мы не углублялись в подробности, касающиеся цифровых подписей. В этом разделе мы рассмотрим, как работают цифровые подписи и каким образом они могут представлять доказательство права владения секретным ключом без предъявления самого секретного ключа.

В биткойн-системе используется алгоритм цифровой подписи *Elliptic Curve Digital Signature Algorithm* (*ECDSA*). ECDSA – это алгоритм создания цифровых подписей на основе пар секретный/открытый ключ, вычисляемых по эллиптическим кривым, как описано в разделе «Криптография с использованием эллиптических кривых» главы 4. Алгоритм ECDSA применяется в скриптовых функциях `OP_CHECKSIG`, `OP_CHECKSIGVERIFY`, `OP_CHECKMULTISIG` и `OP_CHECKMULTISIGVERI-`

FY. Если вы видите эти функции в блокирующем скрипте, то разблокирующий скрипт непременно должен содержать цифровую подпись, созданную с помощью ECDSA.

В биткойн-системе цифровая подпись служит для достижения трех целей (см. примечание-врезку ниже). Во-первых, подпись подтверждает, что владелец секретного ключа, предположительно являющийся владельцем денежных средств, *авторизован* (имеет право) для расходования этих средств. Во-вторых, доказательство авторизации *неоспоримо* (его невозможно отрицать). В-третьих, подпись подтверждает, что рассматриваемая транзакция (или определенные части этой транзакции) не были и *не могли быть изменены* кем-либо после того, как транзакция была подписана.

Отметим, что каждый фрагмент входных данных транзакции подписывается отдельно и независимо от других фрагментов. Это очень важно, так как ни подписи, ни входные данные не обязательно должны принадлежать или применяться одними и теми же «владельцами». На практике этот факт используется специализированной схемой транзакций под названием CoinJoin для создания защищенных транзакций, состоящих из нескольких частей.

Каждый фрагмент входных данных транзакции и каждая подпись в таком фрагменте абсолютно независимы от любого другого фрагмента входных данных или подписи. Для формирования транзакции может использоваться несколько независимых частей, а подписывается каждый фрагмент входных данных по отдельности.

Определение цифровой подписи в Википедии

Электронная подпись (ЭП), Электронная цифровая подпись (ЭЦП), Цифровая подпись (ЦП) – реквизит электронного документа, полученный в результате криптографического преобразования информации с использованием закрытого ключа подписи и позволяющий проверить отсутствие искажения информации в электронном документе с момента формирования подписи (целостность), принадлежность подписи владельцу сертификата ключа подписи (авторство), а в случае успешной проверки подтвердить факт подписания электронного документа (неотказуемость).
Источник: https://ru.wikipedia.org/wiki/Электронная_подпись.
См. также: https://en.wikipedia.org/wiki/Digital_signature.

Как работают цифровые подписи

Цифровая подпись представляет собой математическую схему вычислений (matematical scheme), состоящую из двух частей. Первая часть – алгоритм создания подписи с использованием секретного ключа (ключа подписи) из сообщения (транзакции). Вторая часть – алгоритм, позволяющий любому желающему проверить подлинность подписи, используя для этого также соответствующее сообщение и открытый ключ.

Создание цифровой подписи

В реализации алгоритма ECDSA для биткойн-системы подписываемое «сообщение» представляет собой транзакцию, а если говорить более точно – хэш-значение определенного подмножества данных транзакции (см. раздел «Типы хэш-значений подписи (SIGHASH)» ниже в текущей главе). Ключом для подписи является секретный ключ пользователя. Результатом выполнения алгоритма становится подпись:

$$Sig = F_{sig}(F_{hash}(m), dA),$$

где:

- dA – подписывающий секретный ключ;
- m – транзакция (или ее часть);
- F_{hash} – функция хэширования;
- F_{sig} – алгоритм создания подписи;
- Sig – полученная в результате подпись.

Более подробно математическое обоснование алгоритма ECDSA рассматривается в разделе «Математическое обоснование алгоритма ECDSA» ниже в текущей главе.

Функция F_{sig} генерирует подпись `Sig`, состоящую из двух значений, обычно обозначаемых как `R` и `S`:

```
Sig = (R, S)
```

После вычисления двух значений `R` и `S` выполняется их сериализация в поток байтов по схеме кодирования, определенной международным стандартом *Distinguished Encoding Rules* (*DER*).

Сериализация цифровых подписей (стандарт DER)

Еще раз рассмотрим транзакцию, созданную Алисой. Во входных данных транзакции имеется разблокирующий скрипт, который содержит следующую цифровую подпись кошелька Алисы, закодированную в формате DER:

```
3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b
9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e381301
```

Эта цифровая подпись представляет собой сериализованный поток байтов, составленный из значений `R` и `S`, сгенерированных кошельком Алисы для подтверждения ее права владения секретным ключом, авторизованным для расходования этих выходных данных. Формат сериализации определяет девять элементов, описанных ниже:

- `0x30` – обозначает начало DER-последовательности;
- `0x45` – длина последовательности (69 байтов);
- `0x02` – признак целочисленного значения (см. ниже);
- `0x21` – длина целочисленного значения (33 байта);
- `R` – `00884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb`;
- `0x02` – признак еще одного целочисленного значения (см. ниже);

- `0x20` – длина целочисленного значения (32 байта);
- `S` – `4b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813`;
- `0x01` – суффикс, обозначающий тип используемого хэш-значения (`SIGHASH_ALL`).

Попробуйте декодировать сериализованную подпись Алисы (в кодировке DER), используя приведенный выше список. Здесь наиболее важны числа `R` и `S`, а остальные данные являются частью схемы кодирования по стандарту DER.

Проверка цифровых подписей

Для проверки цифровой подписи необходимо иметь саму подпись (числа R и S), сериализованную транзакцию и открытый ключ (соответствующий секретному ключу, который применялся для создания этой подписи). По существу, проверка подписи означает: «Только владелец секретного ключа, по которому сгенерирован данный открытый ключ, мог создать эту подпись рассматриваемой транзакции».

Алгоритм проверки цифровой подписи принимает как входные данные сообщение (хэш-значение транзакции или ее части), открытый ключ подписавшего и саму подпись (значения `R` и `S`), а возвращает значение TRUE, если подпись подлинная и соответствует рассматриваемому сообщению и открытому ключу.

Типы хэш-значений подписи (SIGHASH)

Цифровые подписи применяются к сообщениям, которые в биткойн-системе представляют собой собственно транзакции. Подпись подразумевает *передачу с подтверждением* (*commitment*) от подписавшего лица некоторого конкретного фрагмента данных в транзакцию. В простейшей форме подпись применяется ко всей транзакции в целом, следовательно, заверяет (подтверждает) все входные и выходные данные и прочие поля транзакции. Но подпись может также заверять лишь некоторое подмножество данных транзакции, что очень удобно во многих случаях, как мы увидим далее в этом разделе.

Подписи в биткойн-системе имеют возможность указывать, какая именно часть данных транзакции включена в хэш-значение, подписанное конкретным секретным ключом. Для этого используется флаг `SIGHASH`, представляющий собой один байт, добавляемый к подписи. В каждой подписи имеется флаг `SIGHASH`, который может быть различным в разных входных данных. Транзакция с тремя подписанными фрагментами входных данных может содержать три подписи с различными флагами `SIGHASH`, при этом каждая подпись заверяет (подтверждает) различные части транзакции.

Следует помнить, что каждый фрагмент входных данных может содержать подпись в своем разблокирующем скрипте. В результате транзакция, содержащая несколько фрагментов входных данных, может включать подписи с различными флагами `SIGHASH`, которые заверяют отдельные части транзакции в каждом фрагменте входных данных. Также отметим, что биткойн-транзак-

ции могут содержать входные данные от различных «владельцев», которые могут подписать только один фрагмент входных данных в частично сформированной (и некорректной) транзакции, поэтому взаимодействуют с другими владельцами, чтобы собрать все необходимые подписи для создания корректной транзакции. Многие из типов флагов SIGHASH имеют смысл только в том случае, если имеется в виду несколько участников, взаимодействующих вне биткойн-сети и вносящих изменения в частично подписанную транзакцию.

Существуют три типа флага SIGHASH: ALL, NONE и SINGLE, – описанных в табл. 6.3.

Таблица 6.3. Типы флага SIGHASH и их описание

Флаг SIGHASH	Значение	Описание
ALL	0x01	Подпись применяется ко всем фрагментам входных и выходных данных
NONE	0x02	Подпись применяется ко всем фрагментам входных данных, но не к фрагментам выходных данных
SINGLE	0x03	Подпись применяется ко всем фрагментам входных данных, но только к одному фрагменту выходных данных с тем же номером индекса, что и подписанные входные данные

Кроме того, существует флаг-модификатор SIGHASH_ANYONECANPAY, который можно объединять с любым из вышеописанных флагов. Когда флаг SIGHASH_ANYONECANPAY установлен, подписывается только один фрагмент входных данных, оставляя прочие фрагменты (и соответствующие им номера последовательностей) открытыми и доступными для внесения изменений. Флаг SIGHASH_ANYONECANPAY имеет значение 0x80 и применяется с помощью оператора побитовой дизъюнкции OR, позволяющего объединять его с флагами SIGHASH, как показано в табл. 6.4.

Таблица 6.4. Типы флага SIGHASH, объединенные с модификатором, и описание результата

Комбинация флагов SIGHASH	Значение	Описание
ALL|ANYONECANPAY	0x81	Подпись применяется только к одному фрагменту входных данных и ко всем фрагментам выходных данных
NONE|ANYONECANPAY	0x82	Подпись применяется только к одному фрагменту входных данных, но не к фрагментам выходных данных
SINGLE|ANYONECANPAY	0x83	Подпись применяется только к одному фрагменту входных данных и фрагменту выходных данных с тем же номером индекса

Способ применения флагов SIGHASH во время процедур подписания и проверки подписей заключается в следующем: создается копия транзакции, и некоторые ее поля усекаются (для них устанавливаются нулевая длина и пустое содержимое). После этого выполняется сериализация усеченной транзакции. Флаг SIGHASH добавляется в конец сериализованной транзакции, и результат хэшируется. Полученное хэш-значение представляет собой подписываемое «сообщение». Выбор различных усекаемых частей транзакции зависит от типа установленного флага SIGHASH, поэтому вычисляемое хэш-значение зависит от

различных подмножеств данных транзакции. Включение флага SIGHASH на последнем шаге перед хэшированием позволяет заверить подписью также и сам тип этого флага, так что его изменение невозможно (например, майнером).

Все типы флага SIGHASH заверяют поле транзакции nLocktime (см. раздел «Время блокировки транзакции (nLocktime)» главы 7). Кроме того, сам тип флага SIGHASH добавляется в транзакцию перед ее подписанием, поэтому тип этого флага невозможно изменить после подписания.

В примере с транзакцией Алисы (см. список в разделе «Сериализация цифровых подписей (стандарт DER)» выше в текущей главе) мы видели, что самой последней частью подписи в кодировке DER было число 01, которое представляет собой флаг SIGHASH_ALL. Он блокирует данные транзакции, поэтому подпись Алисы заверяет состояние всех входных и выходных данных. Это наиболее часто применяемая форма подписи.

Рассмотрим некоторые другие формы и комбинации флагов SIGHASH и возможные варианты их практического применения:

- ALL|ANYONECANPAY – эта комбинация может использоваться для создания транзакции типа «массовый сбор средств». Любой человек, пытающийся собрать некоторую сумму добровольных пожертвований, может сформировать транзакцию с единственным фрагментом выходных данных, в котором указана «целевая сумма» сбора. Очевидно, что такая транзакция изначально некорректна, так как не содержит входных данных. Но другие пользователи теперь могут вносить в нее изменения, добавляя собственные фрагменты входных данных как пожертвования. Транзакция остается некорректной до тех пор, пока не наберется достаточная сумма входных данных, равная значению выходных данных. Каждое пожертвование – это своеобразный «залог», который не может быть принят лицом, организовавшим сбор, до тех пор, пока не будет собрана вся сумма полностью;
- NONE – этот тип флага можно использовать для создания «чека на предъявителя» или «незаполненного чека» на определенную сумму. Заверяются входные данные, но разрешено изменение блокирующего скрипта в выходных данных. Любой может вписать свой биткойн-адрес в блокирующий скрипт выходных данных и перевести транзакцию на себя. Но при этом само значение выходных данных остается заблокированным подписью;
- NONE|ANYONECANPAY – этой комбинацией можно воспользоваться для создания «сборщика мелочи». Пользователи с чрезвычайно малыми данными UTXO в кошельках не могут расходовать их без оплаты транзакций, которая превышает размер этой «мелочи». При использовании этого типа подписи мелкие данные UTXO могут быть переведены на один адрес для объединения и последующего расходования как более приемлемой суммы при необходимости.

Существует много предложений по изменению и усовершенствованию системы флагов SIGHASH. Одно из таких предложений – *Bitmask Sighash Modes* – сделано Гленном Уилленом (Glenn Willen) из Blockstream как часть проекта Elements. Основная идея этого предложения – создать более универсальную замену для типов SIGHASH, которая позволит применять «произвольные, перезаписываемые майнерами битовые маски входных и выходных данных», позволяющие формировать «более сложные контрактные предварительные (перед заверением подписью) схемы, такие как подписанные предложения с возможностью изменения данных на распределенном фондовом рынке активов (имущества)».

Вы не увидите флагов SIGHASH, представленных как некие параметры в приложении кошелька пользователя. Кошельки формируют скрипты P2PKH и подписывают их с флагом SIGHASH_ALL практически всегда, кроме некоторых исключительных случаев. Для применения другого флага SIGHASH необходимо написать программу, формирующую и подписывающую транзакции. И что более важно, флаги SIGHASH могут использоваться специализированными биткойн-приложениями, которые делают доступными новые способы применения этих флагов.

Математическое обоснование алгоритма ECDSA

Как уже было отмечено ранее, подписи создаются с помощью математической функции F_{sig}, которая генерирует подпись, состоящую из двух значений R и S. В этом разделе мы более подробно рассмотрим функцию F_{sig}.

Сначала алгоритм создания подписи генерирует *эфемерную* (*ephemeral*), или временную, пару, состоящую из секретного и открытого ключей. Эта пара временных ключей основана на случайном числе k, которое используется как временный секретный ключ. Из числа k генерируется соответствующий временный открытый ключ P (вычисляемый как $P = k * G$ тем же способом, которым генерируются открытые ключи биткойна; см. раздел «Открытые ключи» главы 4). Затем значением R цифровой подписи становится координата x эфемерного открытого ключа P.

Далее алгоритм вычисляет значение S для цифровой подписи следующим образом:

$$S = k^{-1}(Hash(m) + dA * R) \bmod p,$$

где:

- k – эфемерный секретный ключ;
- R – координата x эфемерного открытого ключа;
- dA – секретный ключ, используемый для данной подписи;
- m – данные транзакции;
- p – простое число – порядок эллиптической кривой.

Проверка представляет собой функцию, обратную функции генерации подписи, и использует значения R, S и открытый ключ для вычисления зна-

чения P, являющегося точкой той же эллиптической кривой (это эфемерный открытый ключ, использованный при создании подписи):

$$P = S^{-1} * Hash(m) * G + S^{-1} * R * Qa,$$

где:

- R и S – значения, составляющие подпись;
- Qa – открытый ключ Алисы;
- m – данные транзакции, которая была подписана;
- G – базовая точка генерации на эллиптической кривой.

Если координата x вычисленной точки P равна R, то проверяющий может утверждать, что проверяемая подпись корректна.

Отметим, что для процедуры проверки цифровой подписи не требуется знание секретного ключа, поэтому открывать его кому-либо нет никакой необходимости.

Математическое обоснование алгоритма ECDSA сложно для понимания. В этой теме помогут разобраться многочисленные подробные руководства, размещенные в Интернете. Выполните поиск по ключевой фразе «описание ECDSA» («ECDSA explained») или сразу воспользуйтесь одним из таких руководств: http://www.morepc.ru/security/crypt/os200207010.html.

Важность фактора случайности в цифровых подписях

В предыдущем разделе «Математическое обоснование алгоритма ECDSA» алгоритм генерации цифровой подписи использовал случайный ключ k как основу для создания эфемерной пары, состоящей из секретного и открытого ключей. Конкретное значение ключа k не важно при условии, что оно действительно является случайным. Если одно и то же значение k используется для генерации двух подписей к различным сообщениям (транзакциям), то секретный ключ, примененный для подписи, может быть вычислен кем угодно. Повторное использование одного и того же значения k в алгоритме создания цифровой подписи приводит к раскрытию секретного ключа, помните об этом всегда.

Если одно и то же значение k используется в алгоритме создания цифровых подписей для двух различных транзакций, то секретный ключ может быть вычислен и станет известным всему миру.

Это не просто теоретическая возможность. Проблемы, приводящие к раскрытию секретных ключей, действительно были обнаружены в нескольких различных реализациях алгоритмов создания подписей транзакций в биткойн-системах. У людей похищали денежные средства из-за неосторожного повторного использования значения k. Наиболее часто встречающейся причиной повторного использования значения k является неправильная инициализация генератора случайных чисел.

Для устранения этой уязвимости существует практическая отраслевая рекомендация: не генерировать k с помощью генератора случайных чисел, инициа-

лизируемого источником какой-либо энтропии, а вместо этого воспользоваться детерминированным случайным процессом, инициализируемым самими данными транзакции. Это обеспечивает получение различных значений *k* для каждой транзакции. Стандартный промышленный алгоритм для детерминированной инициализации *k* определен в документе RFC 6979 (https://tools.ietf.org/html/rfc6979), опубликованном рабочей группой Internet Engineering Task Force (IETF).

Если вы занимаетесь реализацией алгоритма подписи транзакций в биткойн-системе, то обязательно должны использовать стандарт RFC 6979 или аналогичный детерминированный случайный алгоритм для обеспечения генерации различных значений *k* для каждой транзакции.

Биткойн-адреса, балансы и прочие абстракции

В начале текущей главы мы узнали, что внутреннее устройство транзакций существенно отличается от их внешнего представления в кошельках, проводниках блокчейна и прочих приложениях, предоставляющих пользовательский интерфейс. Кажется, что многие упрощенные и хорошо знакомые концепции из предыдущих глав, такие как биткойн-адреса и балансы, совершенно отсутствуют в структуре транзакции. Мы собственными глазами видели, что транзакции не содержат в явном виде биткойн-адреса, вместо этого они функционируют с помощью скриптов, которые блокируют и разблокируют отдельные независимые значения, представляющие суммы в биткойнах. В этой системе балансы не представлены никоим образом, тем не менее каждое приложение кошелька выводит пользовательский баланс в наглядной форме.

Теперь, когда мы узнали, что в действительности содержится в биткойн-транзакции, можно перейти к рассмотрению способов формирования абстракций более высокого уровня, формируемых из простейших компонентов транзакции.

Еще раз взглянем, как транзакция Алисы представлена в широко известном проводнике блокчейна (рис. 6.7).

В левой части представления транзакции проводник блокчейна показывает биткойн-адрес Алисы как «отправителя». В действительности этой информации в самой транзакции нет. При извлечении транзакции проводник блокчейна также восстановил и предыдущую транзакцию, на которую ссылались входные данные, и извлек первый фрагмент выходных данных этой более ранней транзакции. В этом фрагменте выходных данных находится блокирующий скрипт, защищающий данные UTXO с хэш-значением открытого ключа Алисы (скрипт P2PKH). Проводник блокчейна извлекает хэш-значение открытого ключа и выполняет кодирование в формате Base58Check, чтобы получить и вывести биткойн-адрес, представляющий этот открытый ключ.

Похожим образом в правой части проводник блокчейна показывает два фрагмента выходных данных: первый – биткойн-адрес Боба, второй – бит-

койн-адрес Алисы (для получения сдачи). И в этом случае для формирования этих биткойн-адресов проводник блокчейна извлекает блокирующий скрипт из каждого фрагмента выходных данных, определяет его как скрипт P2PKH и получает из него хэш открытого ключа. После этого проводник кодирует полученное значение в формате Base58Check, чтобы сформировать и вывести соответствующие биткойн-адреса.

Рис. 6.7 ❖ Представление транзакции Алисы, выполняющей оплату в кафе Боба

Если щелкнуть по биткойн-адресу Боба, то проводник блокчейна выведет информацию, показанную на рис. 6.8.

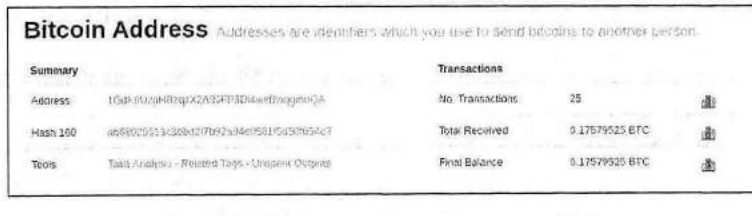


Рис. 6.8 ❖ Баланс биткойн-адреса Боба

Проводник блокчейна выводит баланс биткойн-адреса Боба. Но в биткойн-системе отсутствует такое понятие, как «баланс». Значения, показанные на рис. 6.8, формируются самим проводником блокчейна следующим образом.

Для формирования поля «Total Received» (Всего получено) проводник блокчейна сначала декодирует биткойн-адрес (кодировка Base58Check) для извлечения 160-битового хэш-значения открытого ключа Боба, связанного с этим адресом. Затем проводник выполняет поиск по базе данных транзакций, выбирая все фрагменты выходных данных, которые содержат блокирующий

скрипт P2PKH, содержащий хэш открытого ключа Боба. Просуммировав значения всех найденных выходных данных, проводник блокчейна может вывести общую сумму полученных денежных средств.

Вычисление текущего баланса (поле «Final Balance») требует выполнения несколько большего объема работы. Проводник блокчейна хранит отдельную базу выходных данных, которые на текущий момент пока еще не израсходованы, то есть набор данных UTXO. Для поддержки этой базы данных проводник блокчейна должен в реальном времени наблюдать за всеми операциями в биткойн-сети, добавлять новые созданные данные UTXO, удалять израсходованные данные UTXO сразу после их появления в неподтвержденных транзакциях. Это весьма сложный процесс, зависящий от непрерывного прослеживания продвижения транзакций в сети, а также от обеспечения достижения консенсуса в биткойн-сети, гарантирующего прослеживание корректной цепочки. Иногда проводник блокчейна просто не в состоянии поддерживать синхронизацию, и его представление набора данных UTXO становится неполным или некорректным.

С помощью набора данных UTXO проводник блокчейна суммирует значения всех неизрасходованных выходных данных с хэш-ссылкой на открытый ключ Боба и вычисляет общий текущий баланс (Final Balance), предъявляемый пользователю.

Чтобы сформировать представление с этими двумя «балансами», показанное на рис. 6.8, проводник блокчейна должен проиндексировать и выполнить поиск по десяткам, сотням и даже сотням тысяч транзакций.

В целом информация, предоставляемая пользователям с помощью приложений кошельков, проводников блокчейна и прочих пользовательских интерфейсов биткойна, чаще всего формируется из абстракций более высокого уровня, которые зависят от выполнения процедур поиска по множеству различных транзакций, просмотра их содержимого и обработки данных, включенных в эти транзакции. Предъявляя эти упрощенные представления биткойн-транзакций, напоминающие банковские чеки от одного отправителя одному получателю, приложения должны вывести на требуемый уровень абстракции огромное количество скрытых подробностей. Основное внимание сосредоточено на наиболее часто выполняемых типах транзакций: скрипт P2PKH с подписями SIGHASH_ALL в каждом фрагменте входных данных. Приложения биткойна могут представить более 80% всех транзакций в удобной для чтения форме, но иногда испытывают затруднения при представлении транзакций, отклоняющихся от нормы. Транзакции, содержащие более сложные блокирующие скрипты, или другие флаги SIGHASH, или множество фрагментов входных и выходных данных, наглядно показывают упрощенную сущность и слабые места этих абстракций.

Ежедневно сотни транзакций, не содержащих выходных данных со скриптами P2PKH, подтверждаются и включаются в структуру данных блокчейна. Проводники блокчейна часто представляют такие транзакции с предупреж-

дающими сообщениями красного цвета, информирующими о невозможности декодирования адреса. Следующая ссылка указывает на самую свежую «необычную транзакцию», которую не удалось полностью декодировать: https://blockchain.info/strange-transactions.

В следующей главе мы увидим, что представляют собой такие «необычные транзакции». Это транзакции, содержащие более сложные блокирующие скрипты, чем общепринятый P2PKH. Мы узнаем, как декодировать и понять сложные нестандартные скрипты, и рассмотрим приложения, которые их поддерживают.

Глава 7

Более сложные транзакции и скрипты

Введение

В предыдущей главе мы рассматривали основные элементы биткойн-транзакций и наиболее часто применяемый тип скриптов транзакций – скрипт P2PKH. В этой главе мы будем изучать более изощренные способы создания скриптов и узнаем, как можно использовать их для формирования транзакций со сложными условиями.

Сначала рассматриваются скрипты с мультиподписями (multisignature scripts). Затем особое внимание будет уделено второму по частоте применения скрипту типа Pay-to-Script-Hash, открывающему целый мир сложных скриптов. Далее мы подробно рассмотрим новые операторы скриптов, которые добавляют временны́е характеристики в биткойн-систему с помощью механизма timelock.

Мультиподписи

Скрипты с мультиподписями (multisignature scripts) устанавливают условие, при котором в скрипт записывается N открытых ключей, и должно быть непременно обеспечено наличие не менее M подписей для разблокирования денежных средств. Это также называют схемой M-of-N, где N – общее количество ключей, а M – предельное (пороговое) количество подписей, требуемое для проверки и подтверждения. Например, мультиподпись 2-of-3 содержит три открытых ключа потенциальных авторов подписей, и как ми́нимум два ключа из трех обязательно должны быть использованы при создании подписей для формирования корректной транзакции, расходующей соответствующие денежные средства. В настоящее время в стандартных скриптах с мультиподписями может содержаться до 15 открытых ключей, то есть возможны любые комбинации в диапазоне от 1-of-1 до 15-of-15. Ограничение в 15 ключей может быть рас-

ширено после публикации книги, поэтому используйте функцию `isStandard()`, чтобы узнать текущие приемлемые параметры в своей биткойн-сети.

Общая форма блокирующего скрипта, устанавливающего условие мультиподписи типа M-of-N:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```

где N – общее количество перечисленных открытых ключей, а M – предельное (пороговое) количество требуемых подписей для расходования выходных данных.

Блокирующий скрипт, устанавливающий условие мультиподписи 2-of-3, выглядит следующим образом:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Условие этого блокирующего скрипта может быть выполнено разблокирующим скриптом, содержащим пары подписей, соответствующих перечисленным открытым ключам, например:

```
<Signature B> <Signature C>
```

или любую комбинацию двух подписей, сгенерированных из секретных ключей, соответствующих трем перечисленным выше открытым ключам.

Такие два скрипта вместе образуют объединенный скрипт проверки корректности:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

При выполнении этот объединенный скрипт даст значение `TRUE`, если и только если подписи в разблокирующем скрипте соответствуют условиям, заданным в блокирующем скрипте. В этом случае условие состоит в проверке того факта, что разблокирующий скрипт содержит две корректные подписи, сгенерированные из двух секретных ключей, соответствующих двум из трех открытых ключей, установленных как препятствие.

Ошибка при выполнении оператора CHECKMULTISIG

При выполнении оператора `CHECKMULTISIG` возникает ошибка, которая требует некоторых дополнительных мер по ее устранению (обходу). Оператор `CHECKMULTISIG` должен принимать M+N+2 элемента в стек в качестве параметров. Но из-за ошибки оператор `CHECKMULTISIG` будет извлекать из стека лишнее значение, то есть на одно значение больше, чем предполагается.

Рассмотрим эту ошибку более подробно, воспользовавшись примером проверки из предыдущего раздела:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Сначала оператор `CHECKMULTISIG` извлекает верхний элемент стека, то есть `N` (в рассматриваемом примере это число 3). Затем считывается `N` элементов – открытые ключи, для которых возможны подписи. В рассматриваемом примере это открытые ключи A, B и C. Потом извлекается следующий элемент `M` –

кворум (количество необходимых подписей). Здесь M = 2. После этого оператор CHECKMULTISIG должен извлечь последние M элементов, то есть подписи, и проверить их правильность. Но, к сожалению, из-за ошибки в реализации оператор CHECKMULTISIG считывает один лишний элемент (всего M+1 элементов). При проверке подписей этот лишний элемент не принимается во внимание, поэтому не оказывает прямого воздействия на сам оператор CHECKMULTISIG. Тем не менее такое дополнительное значение обязательно должно находиться в стеке, так как если оно отсутствует, то при попытке извлечь значение из пустого стека возникает ошибка стека и скрипт аварийно завершается (помечая при этом транзакцию как некорректную). Поскольку дополнительное значение не обрабатывается, оно может быть любым, но обычно используется 0.

Так как эта ошибка стала частью правил консенсуса и повторяется всегда, ее необходимо постоянно исправлять. Вот пример исправленного с учетом этой ошибки скрипта проверки подписей:

```
0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Следовательно, и разблокирующий скрипт, в действительности используемый для мультиподписей, должен содержать не просто две подписи:

```
<Signature B> <Signature C>
```

а еще и предшествующий им нуль:

```
0 <Signature B> <Signature C>
```

Теперь, если вы будете рассматривать разблокирующий скрипт для мультиподписей, то вас не должен удивлять дополнительный 0 в начале, единственное назначение которого – обход ошибки, случайно ставшей частью правил консенсуса.

Скрипт Pay-to-Script-Hash (P2SH)

Скрипт Pay-to-Script-Hash (P2SH) появился в 2012 году как новый тип транзакций, расширяющий их возможности и существенно упрощающий использование сложных скриптов транзакций. Чтобы объяснить необходимость скрипта P2SH, рассмотрим практический пример.

В главе 1 мы познакомились с Мохаммедом, импортером бытовой электроники, проживающим в Дубае. Компания Мохаммеда активно использует возможности мультиподписей в биткойн-системе для своих корпоративных учетных записей. Скрипты с мультиподписями – одна из наиболее часто применяемых расширенных функциональных возможностей механизма скриптов, к тому же очень мощная возможность. Компания Мохаммеда использует скрипт с мультиподписями для всех клиентских платежей, в финансовых терминах это обозначается как «дебиторская задолженность» (accounts receivable, AR). При схеме с мультиподписями все платежи клиентов блокируются таким образом, что для разблокировки требуются как минимум две подписи: от Мо-

хаммеда и одного из его партнеров или от его адвоката (юрисконсульта), обладающего резервной копией ключа. Подобная схема с мультиподписями обеспечивает корпоративное управление денежными средствами и защиту от их похищения, незаконной растраты или потери.

Скрипт для этой схемы достаточно длинный и выглядит приблизительно так:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key>
<Attorney Public Key> 5 CHECKMULTISIG
```

Скрипты с мультиподписями обладают мощными функциональными возможностями, но их практическое применение связано с некоторыми трудностями. Располагая показанным выше скриптом, Мохаммед должен передать этот скрипт каждому клиенту до платежа. Каждый клиент должен использовать специализированный биткойн-кошелек с возможностью создания специализированных скриптов транзакций, и каждый клиент обязан знать, как создавать транзакции с помощью таких специализированных скриптов. Кроме того, созданная транзакция будет приблизительно в пять раз больше простой платежной транзакции, потому что специализированный скрипт содержит очень длинные открытые ключи. Издержки по существенно увеличенной транзакции ложатся на клиента в виде ее оплаты. Наконец, скрипт большего размера, подобный приведенному выше, должен быть внесен в набор данных UTXO в оперативной памяти каждого полноценного узла и сохраняться до тех пор, пока не будет израсходован. Все перечисленные проблемы усложняют практическое использование специализированных блокирующих скриптов.

Тип P2SH был разработан для устранения этих практических затруднений, чтобы сделать применение специализированных скриптов таким же простым, как платеж на биткойн-адрес. При платежах типа P2SH сложный блокирующий скрипт заменяется на его цифровые отпечатки, то есть на криптографическое хэш-значение. Если транзакция пытается израсходовать данные UTXO, которые будут представленны позже, то она обязательно должна содержать скрипт, соответствующий хэш-значению, в дополнение к разблокирующему скрипту. Проще говоря, P2SH означает «плата скрипту с совпадающим хэш-значением, а сам скрипт будет представлен позже, когда эти выходные данные будут израсходованы».

В транзакциях типа P2SH блокирующий скрипт, заменяемый своим хэш-значением, называется погашающим скриптом (redeem script), потому что он представлен в системе во время погашения (redemption) существующей задолженности, в отличие от обычного блокирующего скрипта. В табл. 7.1 показан скрипт без использования P2SH, а в табл. 7.2 – тот же скрипт, но закодированный с помощью P2SH.

Таблица 7.1. Сложный скрипт без применения P2SH

Блокирующий скрипт	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Разблокирующий скрипт	Sig1 Sig2

Таблица 7.2. Сложный скрипт в форме P2SH

Погашающий скрипт	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Блокирующий скрипт	HASH160 <20-байтовое хэш-значение погашающего скрипта> EQUAL
Разблокирующий скрипт	Sig1 Sig2 <погашающий скрипт>

В приведенных таблицах можно видеть, что с применением P2SH сложный скрипт, определяющий условия расходования выходных данных (погашающий скрипт), не представлен в блокирующем скрипте. Вместо него в блокирующий скрипт включается только хэш-значение, а сам погашающий скрипт появляется позже, как часть разблокирующего скрипта при расходовании этих выходных данных. Это позволяет перенести издержки, связанные со сложностью транзакции и ее оплатой, с отправителя на получателя транзакции (то есть на того, кто будет расходовать выходные данные).

Рассмотрим, как в компании Мохаммеда используются сложный скрипт с мультиподписями и итоговые скрипты P2SH.

Сначала приводится скрипт с мультиподписями, который компания Мохаммеда применяет для всех входящих платежей от клиентов:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key>
<Attorney Public Key> 5 CHECKMULTISIG
```

Если условные обозначения (шаблоны) заменить на реальные открытые ключи (показанные ниже как 520-битовые числа, начинающиеся с 04), то вы сами можете убедиться в том, что этот скрипт станет очень длинным:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395
D7EEC6984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23
E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D7
8D0E34224858008E8B49047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5
737812F393DA7D4420D7E1A9162F0279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD
7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D0872274
40645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1ECED3C68D446BCAB69AC0BA7DF5
0D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D137AAB59E0B000EB7E
D238F4D800 5 CHECKMULTISIG
```

Весь этот скрипт может быть представлен в виде 20-байтового криптографического хэш-значения, полученного после применения алгоритма хэширования SHA256 и последующего применения к результату алгоритма RIPEMD160. Для приведенного выше скрипта 20-байтовое хэш-значение имеет следующий вид:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

Транзакция типа P2SH блокирует выходные данные по этому хэш-значению вместо более длинного скрипта, используя для этого блокирующий скрипт:

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

который, очевидно, намного короче. Вместо «платы этому скрипту с 5 ключами и мультиподписями» применяется равнозначная транзакция P2SH, то есть

«плата скрипту с заданным хэш-значением». Клиент, выполняющий платеж компании Мохаммеда, должен включить в свою транзакцию только гораздо более короткий блокирующий скрипт. Когда Мохаммед и его партнеры захотят израсходовать эти данные UTXO, они обязаны будут представить исходный погашающий скрипт (хэш-значением которого заблокированы данные UTXO) и подписи, требуемые для разблокировки. Это выглядит приблизительно так:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

Два скрипта объединяются в ходе двух стадий. Сначала погашающий скрипт проверяется на соответствие блокирующему скрипту, то есть должны совпадать хэш-значения:

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160 <redeem scriptHash> EQUAL
```

Если хэш-значения совпадают, то разблокирующий скрипт выполняется отдельно, чтобы разблокировать погашающий скрипт:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```

Почти все скрипты, рассматриваемые в этой главе, могут быть реализованы только как скрипты типа P2SH. Их нельзя использовать непосредственно в блокирующих скриптах для данных UTXO.

Адреса P2SH

Еще одной важной частью механизма P2SH является возможность кодировки скрипта как адреса в соответствии с документом BIP-13. Адреса P2SH – это закодированные в формате Base58Check 20-байтовые хэш-значения скрипта, полностью аналогичные биткойн-адресам, представляющим собой 20-байтовые хэш-значения открытых ключей в кодировке Base58Check. Адреса P2SH используют префикс версии «5», а после кодирования в формате Base58Check получаются адреса, начинающиеся со значения «3». Например, сложный скрипт Мохаммеда после хэширования и кодировки в формате Base58Check как адрес P2SH выглядит так: `39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw`. Теперь Мохаммед может передать этот «адрес» всем своим клиентам, которые получают возможность использования практически любого биткойн-кошелька для выполнения простого платежа, как если бы это был обычный биткойн-адрес. Префикс 3 подсказывает клиентам, что это особый тип адреса, соответствующий скрипту, а не открытому ключу, тем не менее он работает точно так же, как при платеже на биткойн-адрес.

Адреса P2SH позволяют скрыть все сложности процесса, поэтому плательщик освобождается от необходимости разбираться в скрипте.

Преимущества механизма P2SH

Механизм P2SH обладает следующими преимуществами, по сравнению с непосредственным использованием сложных скриптов для блокировки выходных данных:

- сложные скрипты заменяются более короткими цифровыми отпечатками в выходных данных транзакции, уменьшая размер всей транзакции в целом;
- скрипты могут кодироваться как адреса, поэтому получатель и кошелек получателя освобождаются от необходимости использования специализированных механизмов для реализации P2SH;
- механизм P2SH перекладывает обязанность по созданию скрипта на получателя, освобождая плательщика от этой работы;
- механизм P2SH переносит издержки на хранение содержимого длинных скриптов из выходных данных (хранящихся в наборе данных UTXO) во входные данные (хранящиеся в структуре данных блокчейна);
- механизм P2SH переносит издержки на хранение данных для длинных скриптов из текущего времени (время платежа) в будущее (время расходования выходных данных);
- механизм P2SH перекладывает обязанность по оплате транзакции с длинным скриптом с отправителя на получателя, который должен включить в свою транзакцию длинный погашающий скрипт, чтобы расходовать выходные данные.

Погашающий скрипт и проверка корректности

До версии 0.9.2 клиента Bitcoin Core механизм Pay-to-Script-Hash был ограничен стандартными типами скриптов биткойн-транзакций, определяемыми с помощью функции `isStandard()`. Это означает, что погашающий скрипт, представленный в расходной транзакции, должен обязательно принадлежать к одному из стандартных типов: P2PK, P2PKH – или относиться к скриптам с мультиподписями, исключающими возможность применения оператора `RETURN` и собственно механизма P2SH.

В версии 0.9.2 клиента Bitcoin Core транзакции типа P2SH могут содержать любой корректный скрипт, что делает стандарт P2SH более универсальным и позволяет экспериментировать со множеством новых сложных типов транзакций.

Следует отметить, что нельзя поместить скрипт P2SH в погашающий скрипт P2SH, так как спецификация P2SH не допускает рекурсии. Несмотря на наличие технической возможности включения оператора `RETURN` в погашающий скрипт, поскольку нет правил, запрещающих делать это, на практике такое действие не имеет смысла, так как выполнение оператора `RETURN` во время проверки корректности приведет к тому, что транзакция будет помечена как некорректная.

Отметим также, что погашающий скрипт не появляется в сети до попытки расходования выходных данных транзакции типа P2SH, поэтому даже если вы заблокировали выходные данные хэш-значением некорректного погашающего скрипта, он все равно будет обработан. Данные UTXO будут успешно заблокированы. Но в дальнейшем вы не сможете расходовать их, потому что расходная транзакция, включающая этот погашающий скрипт, не будет при-

нята из-за его некорректности. Таким образом, возникает риск, связанный с блокировкой в транзакции типа P2SH биткойнов, которые в дальнейшем невозможно будет потратить. Сеть принимает блокирующий скрипт P2SH, даже если он соответствует некоррректному погашающему скрипту, потому что хэш-значение последнего не дает никакой информации о представляемом им скрипте.

В блокирующие скрипты P2SH включается хэш-значение погашающего скрипта, которое не дает никакой информации о содержимом этого скрипта. Транзакция типа P2SH будет считаться корректной и будет принята в сети, даже если соответствующий погашающий скрипт некорректен. Таким образом, вы можете случайно заблокировать биткойны без возможности их расходования в будущем.

Запись выходных данных (RETURN)

Распределенный и содержащий метки времени реестр биткойн-системы – структура данных блокчейна может применяться не только для учета выполняемых платежей. Многие разработчики пытались использовать язык скриптов транзакций для получения преимуществ в плане безопасности и универсальности этой системы для приложений в таких областях, как цифровые нотариальные службы, акционерные сертификаты (стокноты) и смарт-контракты. При первых попытках применения языка скриптов транзакций для этих целей создавались выходные данные транзакций, записывающие данные в структуру блокчейна. Например, запись цифрового отпечатка некоторого файла таким образом, чтобы любой желающий мог получить доказательство существования (proof-of-existence) этого файла на определенную дату, обратившись к соответствующей транзакции.

Использование структуры данных блокчейна биткойн-системы для хранения данных, не связанных с платежами в биткойнах, является спорной темой, постоянно вызывающей дискусии. Многие разработчики считают такой вариант использования злоупотреблением и отвергают его. Другие смотрят на это как на демонстрацию мощных функциональных возможностей технологии блокчейна и поддерживают подобные эксперименты. Главным аргументом противников использования данных, не связанных с платежами, является «непомерное разрастание структуры блокчейна», перегружающее все активно работающие полноценные узлы блокчейн-системы, вынужденные принять на себя накладные расходы по поддержке дисковой памяти для структуры блокчейна, которая изначально не предназначалась для хранения таких данных. Более того, транзакции такого рода создают данные UTXO, которые невозможно расходовать, используя целевой биткойн-адрес как 20-байтовое поле свободной формы. Так как адрес используется для данных, он не соответствует секретному ключу, в результате конечные данные UTXO вообще невозможно израсходовать – это фальшивый платеж. Таким образом, транзакции, выходные данные которых нельзя расходовать, никогда не будут удалены из набора

данных UTXO, что приводит к непрерывному увеличению размера базы данных UTXO, доводя его до непомерной величины.

В версии 0.9 клиента Bitcoin Core был достигнут определенный компромисс с помощью нового оператора RETURN. Оператор RETURN позволяет разработчикам добавлять 80 байтов данных, не относящихся к платежу, в выходные данные транзакции. Но, в отличие от «фальшивых» данных UTXO, оператор RETURN создает в явной форме «безусловно нерасходуемый» (provably unspendable) элемент выходных данных, которые не требуют сохранения в наборе данных UTXO. Выходные данные оператора RETURN записываются в структуру данных блокчейна, занимают дисковое пространство и увеличивают размер только структуры блокчейна, но не хранятся в наборе данных UTXO, следовательно, не «раздувают» размера пула памяти UTXO и не перегружают полноценных узлов биткойна, освобождая их от необходимости наращивания объема оперативной памяти.

Скрипты, использующие оператор RETURN, выглядят следующим образом:

```
RETURN <data>
```

Объем данных <data> ограничен 80 байтами и чаще всего представляет собой хэш-значение, являющееся результатом работы алгоритма SHA256 (32 байта). Многие приложения размещают перед данными префикс, помогающий идентифицировать тип приложения. Например, цифровая нотариальная служба Proof of Existence (http://proofofexistence.com) использует 8-байтовый префикс DOCPROOF, в ASCII-кодировке выглядящий как 44 4f 43 50 52 4f 4f 46 в шестнадцатеричном формате.

Всегда следует помнить об отсутствии соответствующего оператору RETURN «разблокирующего скрипта», который мог бы «израсходовать» выходные данные RETURN. Самым главным при использовании оператора RETURN является невозможность расходования денег, заблокированных в этом типе выходных данных, таким образом, нет необходимости хранить их в наборе данных UTXO как потенциально расходуемые – данные оператора RETURN безусловно нерасходуемые. Обычно в выходных данных оператора RETURN указывается нулевая сумма биткойнов, потому что любой биткойн, связанный с этим типом выходных данных, будет потерян навсегда. Если на данные RETURN ссылаются как на входные данные транзакции, то механизм проверки корректности скриптов остановит процедуру проверки скрипта и пометит такую транзакцию как некорректную. Выполнение оператора RETURN сразу же приводит к «возврату» из обрабатываемого скрипта со значением FALSE и завершению процесса. Поэтому если вы случайно указали выходные данные RETURN как входные данные следующей транзакции, то эта транзакция некорректна.

Стандартная транзакция (соответствующая условиям, проверяемым функцией isStandard()), может содержать только один фрагмент данных типа RETURN. Тем не менее один фрагмент выходных данных типа RETURN в транзакции может быть объединен с выходными данными любого другого типа.

В версии 0.10 Bitcoin Core добавлены два новых параметра в интерфейс командной строки. Параметр `datacarrier` управляет передачей и майнингом транзакций типа `RETURN` – по умолчанию установлено значение 1, разрешающее эти операции. Для параметра `datacarriersize` устанавливается числовое значение, определяющее максимальный размер в байтах скрипта `RETURN`, по умолчанию равное 83 байтам: максимум 80 байтов данных типа `RETURN`, один байт – код операции `RETURN`, и два байта – код операции `PUSHDATA`.

Изначально для данных типа RETURN было предложено ограничение размера, равное 80 байтам, но этот предел был уменьшен до 40 байтов, когда данная функциональная возможность была реализована. В феврале 2015 года в версии 0.10 Bitcoin Core был возвращен лимит в 80 байтов. Узлы могут выбрать вариант отказа от передачи и майнинга данных типа RETURN или вариант передачи и майнинга только тех данных типа RETURN, которые содержат менее 80 байтов.

Блокировки по времени (timelocks)

Блокировки по времени (timelocks) – это ограничения на транзакции или на выходные данные, позволяющие начать расходование только после определенного момента времени. Биткойн-система с самого начала обладала возможностью блокировок по времени на уровне транзакций, которая реализована с помощью поля `nLocktime` в транзакции. В конце 2015 года и в середине 2016 года появились два новых параметра, определяющих блокировки по времени на уровне данных UTXO: `CHECKLOCKTIMEVERIFY` и `CHECKSEQUENCEVERIFY`.

Блокировки по времени полезны для транзакций, датированных более поздним числом, и для блокировки денежных средств до конкретной даты в будущем. Более важно, что блокировки по времени расширяют возможности биткойн-скриптов, добавляя время как дополнительное измерение, тем самым создавая условия для формирования сложных многоэтапных смарт-контрактов.

Блокирование транзакции по времени (nLocktime)

В биткойн-системе изначально существовала возможность блокировки по времени на уровне транзакции. Блокировка транзакции по времени – это установка параметра уровня транзакции (поле в структуре данных транзакции), который определяет самое раннее время признания транзакции действующей: после наступления указанного времени транзакция может быть передана в сеть или добавлена в структуру блокчейна. Блокировка по времени также известна как `nLocktime` по имени переменной из кодовой базы Bitcoin Core. В большинстве транзакций для этой переменной установлено значение нуль, определяющее немедленное распространение по сети и выполнение. Если переменная `nLocktime` содержит ненулевое значение, меньшее чем 500 миллионов, то это значение интерпретируется как высота блока, то есть транзакция не действительна и не может быть передана в сеть или включена в структуру блок-

чейна до момента достижения заданной высоты блока. Если значение больше 500 миллионов, то оно интерпретируется как метка Unix-времени (количество секунд, прошедших с 1 января 1970 года) и транзакция не будет действительной до наступления заданного времени. Транзакции с ненулевым значением `nLocktime`, определяющим будущий блок или время в будущем, должны быть непременно сохранены операционной системой и переданы в биткойн-сеть только после того, как станут действительными. Если транзакция передается в сеть раньше времени, определяемого полем `nLocktime`, то она отбрасывается первым же узлом как недействительная и не передается на другие узлы. Использование поля `nLocktime` аналогично указанию более поздней даты (в будущем) в бумажном чеке.

Ограничения блокировки по времени на уровне транзакций

Для параметра `nLocktime` имеется ограничение, из-за которого он дает лишь возможность расходования некоторых выходных данных в будущем, но неспособен сделать невозможным их расходование до наступления заданного времени. Рассмотрим это ограничение более подробно на следующем примере.

Алиса подписывает транзакцию, расходующую один из ее фрагментов выходных данных и направляющих этот фрагмент на адрес Боба, и устанавливает значение `nLocktime` равным 3 месяцам в будущем. Алиса отправляет транзакцию Бобу для хранения. При наличии этой транзакции Алиса и Боб знают, что:

- Боб не сможет воспользоваться этой транзакцией для получения в свое распоряжение денежных средств до конца заданного 3-месячного срока;
- по истечении 3-месячного срока Боб сможет воспользоваться этой транзакцией.

Но при этом следует учесть, что:

- Алиса может создать другую транзакцию, используя двойное расходование тех же самых входных данных без блокировки по времени. Таким образом, Алиса может израсходовать те же самые данные UTXO раньше, чем пройдут 3 месяца;
- у Боба нет никакой уверенности в том, что Алиса не сделает этого.

Важно хорошо понимать сущность параметра `nLocktime` для транзакций. Он дает уверенность только в том, что Боб сможет получить деньги из такой транзакции только после окончания установленного 3-месячного срока. Но нет никаких гарантий, что Боб действительно получит эти деньги. Для получения таких гарантий блокировка по времени должна устанавливаться не на транзакцию, а на сами данные UTXO и стать частью блокирующего скрипта. Это достигается с помощью следующей формы блокировки по времени, называемой Check Lock Time Verify.

Check Lock Time Verify (CLTV)

В декабре 2015 года в биткойн-системе появилась новая форма блокировки по времени в виде обновления одной из ветвей программного обеспечения. На

основе спецификаций BIP-65 в язык скриптов был добавлен новый оператор CHECKLOCKTIMEVERIFY (CLTV). Оператор CLTV обеспечивает блокировку по времени только для выходных данных, а не для всей транзакции, как при использовании nLocktime. Это дает гораздо большую гибкость в применении блокировок по времени.

Проще говоря, добавление оператора CLTV в погашающий скрипт позволяет создать ограничение для выходных данных, которые могут быть израсходованы только по истечении заданного срока.

Поле nLocktime обеспечивает блокировку по времени на уровне транзакций. Оператор CLTV предоставляет возможность блокировки только выходных данных.

Оператор CLTV не заменяет параметра nLocktime, он лишь ограничивает конкретные данные UTXO таким образом, что они могут быть израсходованы только в будущей транзакции с параметром nLocktime, для которого установлено большее или равное значение.

Оператор CLTV принимает один параметр в виде числа в том же формате, что и значение для nLocktime (либо высота блока, либо Unix-время). По суффиксу VERIFY можно понять, что CLTV относится к типу операторов, которые останавливают выполнение скрипта, если проверка выходных данных дает значение FALSE. Если при проверке получено значение TRUE, выполнение продолжается.

Для того чтобы заблокировать выходные данные по времени, нужно вставить оператор CLTV в погашающий скрипт для выходных данных транзакции, которая их создает. Например, если Алиса выполняет платеж на адрес Боба, то выходные данные обычно содержат скрипт P2PKH, подобный следующему:

```
DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

Для создания блокировки по времени, например 3 месяца, начиная с текущего момента, транзакция должна сменить тип на P2SH с использованием погашающего скрипта такого вида:

```
<now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

где <now + 3 months> – высота блока или время, равное 3 месяцам и отсчитываемое от момента начала майнинга этой транзакции: текущая высота блока + 12 960 (блоков) или текущее Unix-время + 7 760 000 (секунд). После оператора CHECKLOCKTIMEVERIFY следует оператор DROP – его смысл будет объяснен немного позже.

Когда Боб пытается израсходовать эти данные UTXO, он создает транзакцию, ссылающуюся на эти данные как на входные. Боб использует свою подпись и открытый ключ в разблокирующем скрипте входных данных и устанавливает для транзакции nLocktime большим или равным значению блокировки по времени CHECKLOCKTIMEVERIFY, установленному Алисой. Затем Боб распространяет созданную транзакцию в биткойн-сети.

Транзакция Боба обрабатывается следующим образом. Если параметр CHECKLOCKTIMEVERIFY, установленный Алисой, меньше или равен значению nLocktime

расходующей транзакции, то выполнение скрипта продолжается (как если бы был выполнен оператор NOP – no operation). В противном случае выполнение скрипта прекращается, а транзакция считается некорректной.

Если говорить более точно, параметр CHECKLOCKTIMEVERIFY аварийно останавливает выполнение, помечая транзакцию как некорректную, при следующих условиях (источник: BIP-65):

1. Стек пуст, или
2. Верхний элемент стека меньше нуля, или
3. Тип блокировки по времени (высота блока или метка времени) верхнего элемента стека и поля nLocktime не совпадает, или
4. Верхний элемент стека больше значения поля nLocktime транзакции, или
5. Значение поля nLocktime входных данных равно 0xffffffff.

Параметры CLTV и nLocktime используют одинаковый формат для описания блокировки по времени: высоту блока или Unix-время в секундах. При совместном использовании этих параметров очень важно, чтобы формат поля nLocktime обязательно совпадал с форматом параметра CLTV во входных данных – оба параметра должны содержать либо высоту блока, либо время в секундах.

После выполнения, если условия, определяемые параметром CLTV, выполнены, предыдущий параметр времени остается в стеке как верхний элемент и, возможно, должен быть удален с помощью оператора DROP для правильного выполнения последующих операторов скрипта. Поэтому вы часто будете встречать в скриптах оператор CHECKLOCKTIMEVERIFY, за которым следует оператор DROP.

При использовании nLocktime в сочетании с CLTV сценарий, описанный в разделе «Ограничения блокировки по времени на уровне транзакций» выше в этой главе, изменяется. Так как Алиса блокирует только данные UTXO, теперь невозможно расходование этих данных ни Бобом, ни Алисой до завершения 3-месячного срока блокировки.

С внесением функциональности блокировки по времени непосредственно в язык скриптов оператор CLTV позволяет разрабатывать некоторые весьма интересные сложные скрипты.

Стандарт определен в документе BIP-65 (CHECKLOCKTIMEVERIFY) (https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki).

Относительные блокировки по времени

Поле nLocktime и параметр CLTV являются абсолютными блокировками (absolute timelocks) по времени, так как определяют абсолютный момент времени. Далее мы будем рассматривать два типа блокировок, являющихся относительными блокировками по времени (relative timelocks), поскольку они устанавливают условие расходования выходных данных как время, прошедшее с момента подтверждения приема выходных данных в структуру блокчейна.

Относительные блокировки по времени удобны, так как позволяют создать цепочку из двух и более взаимосвязанных транзакций и сохранять ее благода-

ря тому, что ограничение по времени для одной транзакции зависит от времени, прошедшего с момента подтверждения предыдущей транзакции. Другими словами, отсчет времени не начнется, пока данные UTXO не будут записаны в структуру блокчейна. Эта функциональность особенно удобна в двунаправленных каналах состояний и в сети Lightning Networks, как мы увидим в разделе «Каналы платежей и каналы состояний» в главе 12.

Относительные блокировки по времени, как и абсолютные блокировки по времени, реализованы и на уровне транзакций, и в виде операций на уровне скриптов. Относительная блокировка по времени на уровне транзакции реализована как правило консенсуса по значению nSequence – поля, устанавливаемого во входных данных каждой транзакции. На уровне скриптов относительные блокировки по времени реализованы в форме кода операций CHECKSEQUENCEVERIFY (CSV).

Реализация относительных блокировок по времени выполнена в соответствии со спецификациями BIP-68 Relative lock-time using consensus-enforced sequence numbers (https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki) и BIP-112 CHECKSEQUENCEVERIFY (https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki).

Документы BIP-68 и BIP-112 введены в действие в мае 2016 года как обновление ветви программного обеспечения для правил консенсуса.

Относительные блокировки по времени, устанавливаемые полем nSequence

Относительные блокировки по времени могут быть установлены в каждом фрагменте входных данных транзакции с помощью поля nSequence.

Изначальный смысл поля nSequence

Изначально поле nSequence предназначалось (но никогда не было реализовано соответствующим образом) для обеспечения возможности изменения транзакций в пуле памяти (mempool). При таком способе использования транзакции, содержащие входные данные со значением поля nSequence, меньшим 2^{32} (0xFFFFFFFF), считались «пока еще не завершенными полностью». Такая транзакция должна была сохраняться в пуле памяти до тех пор, пока ее не заменят на другую транзакцию, расходующую те же самые входные данные, с бóльшим значением поля nSequence. После приема транзакции, входные данные которой содержали поле nSequence со значением 2^{32}, она считалась «полностью завершенной» (finalized) и становилась доступной для майнинга.

Поле nSequence никогда не было реализовано должным образом для этой исходной цели, поэтому значение поля nSequence обычно устанавливается равным 2^{32} в транзакциях, которые не используют блокировок по времени. Для транзакций, использующих поле nLocktime или параметр CHECKLOCKTIMEVERIFY, значение поля nSequence обязательно должно быть меньше 2^{32}, чтобы блокировка сработала. В таких случаях обычно устанавливается значение $2^{32} - 1$ (0xFFFFFFFE).

Поле nSequence как относительная блокировка по времени, определяемая правилами консенсуса

После ввода в действие документа BIP-68 новые правила консенсуса применяются для любой транзакции, содержащей входные данные со значением поля nSequence, меньшим 2^{31} (бит 1<<31 не установлен). Это означает, что если самый старший значащий бит (бит 1<<31) не установлен, то получен флаг «относительной блокировки по времени». В противном случае (бит 1<<31 установлен) поле nSequence резервируется для других целей, таких как разрешение CHECKLOCKTIMEVERIFY, nLocktime, Opt-In-Replace-By-Fee, и для прочих будущих разработок.

Транзакции с входными данными, в которых значение поля nSequence меньше 2^{31}, определяются как транзакции с установленной относительной блокировкой по времени. Такая транзакция становится корректной только после достижения входными данными «возраста», установленного значением относительной блокировки. Например, транзакция, во входных данных которой установлена относительная блокировка по времени с полем nSequence, равным 30 блокам, станет корректной, когда будет завершен майнинг как минимум 30 блоков с момента создания ссылки на эти входные данные как на данные UTXO. Так как поле nSequence имеется в каждом фрагменте входных данных, транзакция может содержать любое количество фрагментов входных данных, заблокированных по времени, а чтобы вся транзакция стала корректной, все фрагменты обязательно должны достичь заданного «возраста». В транзакцию можно включать и фрагменты с блокировкой по времени (nSequence < 2^{31}), и фрагменты без относительной блокировки по времени (nSequence >= 2^{31}).

Значение поля nSequence определяется в блоках или в секундах, но в формате, немного отличающемся от формата, используемого для поля nLocktime. Флаг типа (type-flag) предназначен для того, чтобы различать значения, устанавливающие высоту блока, и значения времени в секундах. Флаг типа устанавливается в 23-м наименьшем значащем бите (то есть значение 1<<22). Если флаг типа установлен, то значение поля nSequence интерпретируется как множитель для 512 секунд. Если флаг типа не установлен, то значение поля nSequence интерпретируется как количество блоков.

При обработке поля nSequence как относительной блокировки по времени рассматриваются только 16 младших битов. После вычисления флагов (биты 32 и 23) значение nSequence обычно маскируется с помощью 16-битовой маски (например, nSequence & 0x0000FFFF).

На рис. 7.1 показана схема битовой структуры значения nSequence в соответствии с ее определением в документе BIP-68.

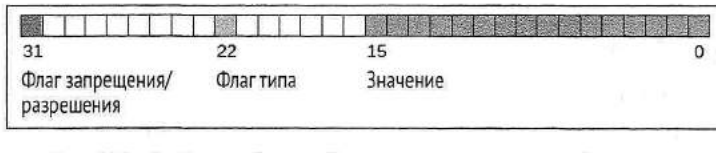


Рис. 7.1 ❖ Схема битовой структуры значения nSequence (источник: BIP-68)

Относительные блокировки по времени основаны на применении правил консенсуса с использованием значения поля nSequence, определенного в документе BIP-68.

Стандарт определен в документе BIP-68 Relative lock-time using consensus-enforced sequence numbers (https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki).

Относительные блокировки по времени с применением параметра CSV

По аналогии с CLTV и nLocktime для скриптов существует код операции для относительных блокировок по времени, позволяющий работать со значением nSequence в скриптах. Это операция CHECKSEQUENCEVERIFY, для краткости чаще обозначаемая кодом CSV.

Код операции CSV при вычислениях в погашающем скрипте данных UTXO позволяет расходование только в тех транзакциях, в которых значение nSequence больше или равно значению параметра CSV. По существу, это ограничивает расходование данных UTXO, запрещая его до достижения заданной высоты блока (заданного количества блоков) или по прошествии заданного времени в секундах относительно момента завершения майнинга этих данных UTXO.

Как и в случае использования CLTV, значение CSV обязательно должно иметь тот же формат, что и соответствующее значение поля nSequence. Если значение CSV задано в блоках, то и значение nSequence должно быть задано в блоках. Если значение CSV задано в секундах, то и значение nSequence должно быть задано в секундах.

Относительные блокировки по времени с использованием операции CSV особенно удобны и полезны при создании нескольких (объединенных в цепочку) транзакций, которые подписываются, но не распространяются в сети, пока они «замкнуты в цепочке». Транзакцию-потомок невозможно использовать до тех пор, пока транзакция-родитель не будет распространена в сети, не пройдет процедуру майнинга и не достигнет «возраста», определенного относительной блокировкой. Один из примеров приложения с использованием относительной блокировки по времени можно будет увидеть в разделе «Каналы платежей и каналы состояний» в разделе «Маршрутизируемые каналы платежей (Lightning Network)» в главе 12.

Подробное определение и описание операции CSV приведены в документе BIP-112 CHECKSEQUENCEVERIFY (https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki).

Median-Time-Past

Частью процедуры активации относительных блокировок также является возможность выбора способа вычисления «времени» блокировок (как абсолютных, так и относительных). В биткойн-системе существует трудноуловимое, но чрезвычайно важное различие между общим временем выполнения (wall time)

и временем достижения консенсуса (consensus time). Биткойн – децентрализованная сеть, поэтому каждый ее участник ведет собственный отсчет времени. События в сети не происходят мгновенно и одновременно во всех ее узлах. Сетевые задержки обязательно должны приниматься во внимание при расчетах времени на каждом узле. В конечном итоге события синхронизируются для создания общего реестра. Биткойн-сеть достигает консенсуса каждые 10 минут, поддерживая текущее состояние реестра, существующее в «недавнем прошлом».

Метки времени, размещаемые в заголовках блоков, устанавливаются майнерами. При этом предполагается некоторая степень свободы действий, допускаемая правилами консенсуса при учете различий в точности отсчета времени между различными узлами. Но это создает для майнеров нездоровый соблазн дать ложные данные о времени в блоке, чтобы получить дополнительные отчисления при включении в блок заблокированных по времени транзакций, которые еще не достигли требуемого «возраста». Более подробно об этой проблеме см. следующий раздел.

Чтобы устранить стимул к передаче ложной информации и увеличить надежность защиты блокировок по времени, был предложен документ BIP, быстро введенный в действие в то же время, что и документы BIP для относительных блокировок по времени. Это документ BIP-113, определяющий новую методику измерения времени достижения консенсуса, названную Median-Time-Past.

По этой методике время вычисляется по меткам времени, взятым из 11 последних блоков, и в этом наборе находится медиана (срединное значение). Затем это медианное время становится временем консенсуса и используется для всех расчетов блокировок по времени. Принятие срединного значения за период около двух часов в прошлом позволяет уменьшить влияние любого значения метки времени из отдельного блока. При обработке набора из 11 блоков ни один из майнеров не может повлиять на значение метки времени с целью получения оплаты за транзакции с блокировками, время которых еще не истекло.

Методика Median-Time-Past изменяет реализацию способов вычисления времени для `nLocktime`, `CLTV`, `nSequence` и `CSV`. Время консенсуса, вычисляемое по методике Median-Time-Past, всегда отстает приблизительно на один час от текущего времени. Если вы создаете транзакции с блокировками по времени, то следует учесть это отставание при оценке требуемого значения, кодируемого в параметрах (полях) `nLocktime`, `nSequence`, `CLTV` и `CSV`.

Методика Median-Time-Past определена в документе BIP-113 (https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki).

Защита блокировок по времени от нелегального получения отчислений

Нелегальное получение оплаты за транзакции (fee-sniping) – это теоретически возможный сценарий атаки, при котором майнеры пытаются переписать последние (самые свежие) блоки и «крадут» транзакции с большими суммами оплаты из будущих блоков для получения максимальной прибыли.

Например, предположим, что самый «высокий» блок в существующей цепочке имеет номер #100000. Вместо попытки майнинга очередного блока #100001 для наращивания цепочки некоторые майнеры пытаются повторно выполнить майнинг блока #100000. Эти майнеры могут выбрать любую корректную транзакцию (которая еще не была вовлечена в процесс майнинга) для включения в свой блок-кандидат #100000. Они не обязаны выполнять повторный майнинг блока с теми же транзакциями. В действительности для майнеров-обманщиков существует стимул для выбора наиболее выгодных (с самой крупной оплатой за Кб) транзакций, чтобы включить их в свой блок. Они могут включить любые транзакции, которые содержались в «старом» блоке #100000, а также любые транзакции из текущего пула транзакций. В итоге при повторном создании блока #100000 эти майнеры получают возможность перемещать транзакции из «настоящего» в переписываемое ими «прошлое».

В настоящее время атака такого типа не очень прибыльна, потому что вознаграждение за новый блок намного выше, чем суммарная оплата любого отдельного блока. Но в некоторый момент в будущем оплата транзакций станет основной частью вознаграждения (или даже полной суммой вознаграждения). Тогда описанной выше ситуации не избежать.

Для защиты от «нелегального получения оплаты» при создании транзакций Bitcoin Core используется поле `nLocktime` со значением по умолчанию «до следующего блока». В описанном выше сценарии Bitcoin Core должен установить значение `nLocktime` равным 100001 в любой создаваемой транзакции. При обычных условиях это значение `nLocktime` не оказывает никакого воздействия – транзакции в любом случае могут быть включены только в блок #100001, так как это следующий блок.

Но при атаке, нарушающей структуру блокчейна, майнеры не должны получать возможности извлекать высокооплачиваемые транзакции из пула памяти, поскольку предполагается, что все эти транзакции заблокированы по времени для блока #100001. Майнеры могут выполнить повторный майнинг блока #100000 только с теми транзакциями, которые были признаны легальными на момент его создания и, по существу, не получают никаких новых отчислений.

Для достижения этой цели Bitcoin Core для всех новых транзакций устанавливает значение поля `nLocktime` равным <номер_текущего_блока + 1>, а значение поля `nSequence` – равным `0xFFFFFFFE`, чтобы разрешить использование `nLocktime`.

Скрипты с управлением потоком выполнения (условные выражения)

Одной из наиболее мощных функциональных возможностей языка Bitcoin Script является управление потоком выполнения, известное также как условные выражения. Возможно, вам знакомы средства управления потоком выполнения в различных языках программирования, использующих конструкцию

IF...THEN...ELSE. Условные выражения биткойна выглядят немного по-другому, но, по существу, это та же самая конструкция.

При самом простом способе применения коды условных операций позволяют создать погашающий скрипт с двумя ветвями выполнения при разблокировке в зависимости от итогового значения TRUE/FALSE, получаемого при вычислении логического условия. Например, если x имеет значение TRUE, то выполняется погашающий скрипт A, иначе (ELSE) выполняется погашающий скрипт B.

Кроме того, условные выражения биткойна могут быть неограниченно вложенными, то есть внутри условного выражения может быть размещено другое условное выражение, которое также содержит следующее условное выражение, и т. д. Управление потоком выполнения Bitcoin Core можно использовать для создания очень сложных скриптов с сотнями и даже тысячами возможных путей выполнения. Уровень вложений не ограничен, но правила консенсуса ограничивают максимальный размер скрипта в байтах.

Управление потоком выполнения биткойна реализовано с использованием кодов операций IF, ELSE, ENDIF и NOTIF. Кроме того, условные выражения могут содержать логические операторы, такие как BOOLAND, BOOLOR и NOT.

На первый взгляд, скрипты с использованием средств управления потоком выполнения биткойна могут показаться запутанными. Причина этого заключается в том, что Bitcoin Script является стековым языком. Точно так же, как выражение 1 + 1 выглядит «вывернутым наизнанку» при записи в формате 1 1 ADD, операторы управления потоком выполнения в биткойн-системе также выглядят записанными «задом наперед».

В большинстве распространенных (процедурных) языков программирования управление потоком выполнения выглядит следующим образом:

```
if (condition):
  code to run when condition is true
else:
  code to run when condition is false
code to run in either case
```

В стековом языке, подобном Bitcoin Script, логическое условие предшествует оператору IF, поэтому выглядит записанным «в обратном порядке»:

```
condition
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

При чтении кода Bitcoin Script следует помнить, что вычисляемое логическое условие всегда записывается перед оператором IF.

Условные выражения с применением оператора VERIFY

Другой формой условных выражений в языке Bitcoin Script является любая операция, заканчивающаяся ключевым словом VERIFY. Суффикс VERIFY означает, что если вычисляемое условие не дает результата TRUE, то выполнение скрипта немедленно завершается, а транзакция объявляется некорректной.

В отличие от оператора IF, который предлагает альтернативные пути выполнения, суффикс VERIFY работает как оператор защиты (guard clause), позволяющий продолжить выполнение только в том случае, если предварительное условие выполнено.

Например, в следующем скрипте требуются подпись Боба и предварительно подготовленный (секретный) образ данных, по которому вычисляется специальное хэш-значение. Для разблокировки должны быть обязательно выполнены оба условия:

```
HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey> CHECKSIG
```

Чтобы погасить (израсходовать) эти денежные средства, Боб должен создать разблокирующий скрипт, в котором представлены правильный предварительно подготовленный образ данных и подпись:

```
<Bob's Sig> <hash pre-image>
```

Без правильного предварительно подготовленного образа данных Боб просто не сможет получить доступ к той части скрипта, в которой проверяется подлинность его подписи.

Тот же скрипт можно написать по-другому, с использованием оператора IF:

```
HASH160 <expected hash> EQUAL
IF
   <Bob's Pubkey> CHECKSIG
ENDIF
```

Разблокирующий скрипт Боба остается тем же самым:

```
<Bob's Sig> <hash pre-image>
```

Скрипт с использованием оператора IF выполняет ту же функцию, что и код с суффиксом VERIFY, – оба варианта работают как защитные условные выражения. Тем не менее конструкция с использованием VERIFY более эффективна, так как в ней на один оператор меньше.

Когда следует использовать оператор VERIFY, а когда оператор IF? Если необходимо всего лишь добавить предварительное (защитное) условие, то лучше воспользоваться оператором VERIFY. Но если нужно определить несколько вариантов выполнения (управление потоком выполнения), то потребуется конструкция IF...ELSE.

Оператор, подобный EQUAL, записывает результат (TRUE/FALSE) в стек, делая его доступным для вычисления с помощью последующих операторов. В противоположность ему оператор-суффикс EQUALVERIFY ничего не оставляет в стеке. Команды, завершающиеся суффиксом VERIFY, не записывают результат в стек.

Использование средств управления потоком выполнения в скриптах

Наиболее часто применяемый вариант использования средств управления потоком выполнения языка Bitcoin Script – создание погашающего скрипта, который предлагает несколько ветвей выполнения, соответствующих различным способам погашения данных UTXO.

Рассмотрим простой пример, в котором участвуют две стороны, подписывающие транзакции, Алиса и Боб, при этом любой из них имеет возможность погашения. При использовании механизма мультиподписи это может быть оформлено как скрипт с мультиподписями типа 1-of-2. В учебных целях для демонстрации средств управления потоком выполнения мы сделаем то же самое с помощью оператора IF:

```
IF
 <Alice's Pubkey> CHECKSIG
ELSE
 <Bob's Pubkey> CHECKSIG
ENDIF
```

Возможно, вы очень удивитесь, взглянув на этот скрипт: «А где же условие? Перед оператором IF ничего нет».

Условие не является частью погашающего скрипта. Вместо этого условие будет предложено в разблокирующем скрипте, позволяя Алисе и Бобу «выбрать» нужную ветвь выполнения.

Алиса осуществляет погашение с помощью следующего разблокирующего скрипта:

```
<Alice's Sig> 1
```

Значение 1 в конце служит условием (TRUE), позволяющим выполнить в приведенной выше конструкции IF первую ветвь погашения, для которой предоставлена подпись Алисы.

Чтобы погашение мог выполнить Боб, он должен обеспечить выбор второй ветви выполнения, передавая значение FALSE в конструкцию IF:

```
<Bob's Sig> 0
```

Разблокирующий скрипт Боба помещает 0 в стек, заставляя конструкцию IF выполнить второй вариант (ELSE) скрипта, для которого требуется подпись Боба.

Поскольку операторы IF могут быть вложенными, есть возможность создать целый «лабиринт» из возможных ветвей выполнения. Разблокирующий скрипт может предоставить «карту» выбора варианта для действительного выполнения:

```
IF
  script A
ELSE
  IF
```

```
    script B
  ELSE
    script C
  ENDIF
ENDIF
```

В этом сценарии три возможные ветви выполнения (`script A`, `script B` и `script C`). Разблокирующий скрипт обеспечивает выбор нужной ветви в форме последовательности значений `TRUE` или `FALSE`. Например, для выбора варианта `script B` разблокирующий скрипт должен заканчиваться числами `1 0` (`TRUE`, `FALSE`). Эти значения будут записаны в стек, и второе значение (`FALSE`) окажется на вершине стека. Внешний оператор `IF` извлекает значение `FALSE` из стека и выполняет первую ветвь `ELSE`. После этого значение `TRUE` перемещается на вершину стека и берется для вычисления внутренним (вложенным) оператором `IF`, который выбирает ветвь выполнения `B`.

Используя эту конструкцию, можно формировать погашающие скрипты с десятками и даже сотнями ветвей выполнения, каждый из которых предлагает собственный способ погашения данных UTXO. Для расходования формируется разблокирующий скрипт, обеспечивающий выбор требуемой ветви выполнения, размещая соответствующие значения `TRUE` и `FALSE` в стеке для управления каждым пунктом разветвления потока выполнения.

Пример сложного скрипта

В этом разделе мы объединим большинство концепций, рассмотренных в данной главе, в едином примере.

Пример использует историю Мохаммеда, владельца компании в Дубаи, занимающегося бизнесом, связанным с импортом/экспортом товаров.

В этом примере Мохаммеду необходимо создать главную учетную запись (счет) для всей компании с гибкими правилами. При такой схеме требуются различные уровни авторизации, зависящие от блокировок по времени. Участниками схемы с мультиподписями являются Мохаммед, два его партнера – Саед и Заира, а также юрисконсульт компании Абдул. Три партнера принимают решения на основе правила большинства, то есть два из трех должны прийти к соглашению. Но в случае возникновения проблемы с их ключами возникает необходимость предоставить юрисконсульту компании возможность восстановления денежных средств в сочетании с подписью одного из трех партнеров. Наконец, если все партнеры недоступны или недееспособны в течение некоторого времени, то они хотели бы передать юрисконсульту право непосредственного управления учетной записью (счетом).

Ниже приведен скрипт, который Мохаммед сформировал для достижения описанных выше целей:

```
IF
  IF
    2
```

```
  ELSE
    <30 days> CHECKSEQUENCEVERIFY DROP
    <Abdul the Lawyer's Pubkey> CHECKSIGVERIFY
    1
  ENDIF
  <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's Pubkey> 3 CHECKMULTISIG
ELSE
  <90 days> CHECKSEQUENCEVERIFY DROP
  <Abdul the Lawyer's Pubkey> CHECKSIG
ENDIF
```

В скрипте Мохаммеда реализованы три ветви выполнения с использованием вложенных конструкций управления потоком IF...ELSE.

В первой ветви выполнения этот скрипт работает как простая схема мультиподписей типа 2-of-3 с тремя партнерами. Эта ветвь выполнения расположена в строках 3 и 9. В строке 3 устанавливается кворум мультиподписей, равный 2 (2 из 3). Эту ветвь выполнения можно выбрать, разместив значения TRUE TRUE в конце разблокирующего скрипта:

```
0 <Mohammed's Sig> <Zaira's Sig> TRUE TRUE
```

Значение 0 в начале этого разблокирующего скрипта помещено из-за ошибки в операторе CHECKMULTISIG, который извлекает лишнее значение из стека. Это значение не учитывается оператором CHECKMULTISIG, но обязательно должно присутствовать, чтобы скрипт не завершился аварийно. Запись (произвольного) значения 0 в стек – это обход ошибки, описанный в разделе «Ошибка при выполнении оператора CHECKMULTISIG» ранее в текущей главе.

Вторая ветвь выполнения может использоваться только по истечении 30-дневного срока после создания данных UTXO. После завершения этого срока требуется подпись юрисконсульта Абдула и одного из трех партнеров (мультиподпись типа 1-of-3). Это реализовано в строке 7, где устанавливается кворум для мультиподписей, равный 1. Для выбора этой ветви выполнения разблокирующий скрипт должен заканчиваться значениями FALSE TRUE:

```
0 <Saeed's Sig> <Abdul's Sig> FALSE TRUE
```

Но почему FALSE TRUE? Разве не в обратном порядке? Потому что эти два значения записываются в стек, причем FALSE записывается первым, затем TRUE вторым. Следовательно, значение TRUE будет извлечено первым при выполнении первого оператора IF.

Третья ветвь выполнения позволяет одному юрисконсульту Абдулу расходовать денежные средства, но только по прошествии 90 дней. Для выбора этой ветви выполнения в конце разблокирующего скрипта должно быть помещено значение FALSE:

```
<Abdul's Sig> FALSE
```

Попробуйте «прогнать» этот скрипт на бумаге, чтобы увидеть, как он работает со стеком.

И еще несколько заданий, связанных с изучением этого скрипта. Сможете ли вы найти ответы на следующие вопросы:

- почему юрисконсульт не сможет выполнить погашения по третьей ветви выполнения в любое время, поместив значение FALSE в разблокирующий скрипт?
- сколько ветвей выполнения может быть использовано для 5, 35 и 105 дней соответственно после завершения майнинга данных UTXO?
- будут ли безвозвратно потеряны денежные средства, если юрисконсульт потеряет свой ключ? Изменится ли ваш ответ, если прошел уже 91 день?
- как партнеры могут «восстанавливать» отсчет времени каждые 29 или 89 дней, чтобы предотвратить единоличный доступ юрисконсульта к денежным средствам?
- почему некоторые операторы CHECKSIG в этом скрипте содержат суффикс VERIFY, в то время как в других этот суффикс отсутствует?

Глава 8

Сеть биткойна

Архитектура пиринговой сети

Структура биткойн-системы представляет собой архитектуру пиринговой сети (peer-to-peer network), сформированную «поверх» глобальной сети Интернет. Сам термин «пиринговый» (peer-to-peer, P2P) означает, что компьютеры в такой сети являются равнозначными партнерами с одинаковыми правами, здесь нет «особенных» узлов, все узлы в одинаковой мере обеспечивают функциональность сетевых сервисов. Соединения между узлами осуществляются по схеме ячеистой сети (mesh network) с «плоской» топологией. В такой сети нет сервера, нет централизованных сервисов, отсутствует иерархия. Узлы в пиринговой сети одновременно и предоставляют сервисы, и пользуются ими, и такое взаимодействие становится стимулом для участия. Пиринговые сети по своей природе гибки, децентрализованы и открыты. Хорошим примером пиринговой сетевой архитектуры являлась сеть Интернет на ранних стадиях своего развития, когда узлы в IP-сети были равноправными. Сегодня архитектура Интернета стала в большей степени иерархической, но межсетевой протокол Internet Protocol (IP) сохраняет свою сущность – простая плоская топология. До появления биткойна самым крупным и успешным практическим приложением пиринговых технологий были системы совместного использования файлов, первой из которых появилась сеть Napster, затем система BitTorrent как самый свежий пример развития этой архитектуры.

Пиринговая сетевая архитектура биткойна – это нечто гораздо большее, чем просто выбор топологии. Биткойн по своей сущности представляет собой пиринговую цифровую систему платежей, а сетевая архитектура одновременно является и отражением, и основанием этой главной характеристики. Децентрализация управления – главный теоретический принцип, реализовать и поддерживать который можно только с помощью простой (плоской) децентрализованной пиринговой консенсусной сети.

Термин биткойн-сеть, или сеть биткойна (bitcoin network), обозначает набор узлов, работающих по пиринговому протоколу биткойна (bitcoin P2P protocol). В дополнение к пиринговому протоколу биткойна используются другие про-

токолы, например Stratum, применяемый для майнинга, упрощенных реализаций или для мобильных кошельков. Работа по дополнительным протоколам поддерживается шлюзами-маршрутизаторами, которые обеспечивают доступ к биткойн-сети с использованием пирингового протокола биткойна и доступ к узлам, работающим по другим протоколам. Например, серверы Stratum устанавливают соединения узлов майнинга по протоколу Stratum с основной биткойн-сетью и являются связующим звеном между протоколом Stratum и пиринговым протоколом биткойна. Мы используем термин «расширенная биткойн-сеть» (extended bitcoin network) для обозначения всей сети в целом, включающей пиринговый протокол биткойна, протоколы пулов майнинга, протокол Stratum и все прочие протоколы, обеспечивающие соединение компонентов биткойн-системы.

Типы и роли узлов

Несмотря на то что в пиринговой сети биткойна все узлы равны, они могут выступать в различных ролях в зависимости от функциональности, которую они предоставляют в текущий момент. Биткойн-узел (bitcoin node) – это набор функций: маршрутизация, база данных блокчейна, майнинг и сервисы кошельков. Полноценный узел с поддержкой всех четырех перечисленных функций показан на рис. 8.1.

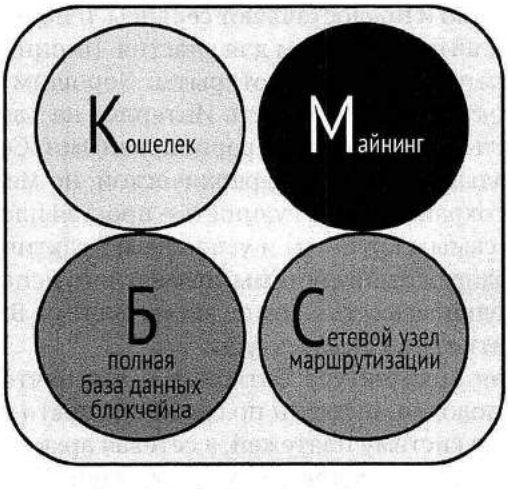


Рис. 8.1 ❖ Узел биткойн-сети с поддержкой всех четырех функций: кошелек, майнинг, полная база данных блокчейна и сетевая маршрутизация

Все узлы поддерживают функцию маршрутизации, необходимую для участия в биткойн-сети и обеспечения возможности поддержки других функций.

Все узлы проверяют корректность и распространяют транзакции и блоки, осуществляют обнаружение и поддержку соединений с партнерами. В примере полноценного узла на рис. 8.1 функция маршрутизации обозначена оранжевым кругом с надписью «Сетевой узел маршрутизации» или просто буквой «С».

Некоторые узлы, называемые полноценными узлами (full nodes), также поддерживают полную и актуальную копию структуры данных блокчейна. Полноценные узлы способны независимо и авторитетно проверить и подтвердить корректность любой транзакции без каких-либо внешних ссылок и обращений. Некоторые узлы поддерживают лишь определенное подмножество структуры данных блокчейна и проверяют транзакции, применяя метод под названием упрощенная верификация платежей (simplified payment verification, SPV). Эти узлы так и называют – SPV-узлы, или упрощенные (lightweight) узлы. В примере полноценного узла на рис. 8.1 функция поддержки полной копии структуры данных блокчейна обозначена синим кругом с надписью «Полная база данных блокчейна» или просто с буквой «Б». На рис. 8.3 SPV-узлы изображены без этого синего круга, показывая, что они не содержат полной копии базы данных блокчейна.

Узлы майнинга (mining nodes) конкурируют в процессе создания новых блоков, используя специализированные аппаратные средства для решения задачи по алгоритму доказательства выполнения работы (Proof-of-Work). Некоторые узлы майнинга являются также полноценными узлами, поддерживающими полную копию базы данных блокчейна, другие – упрощенным узлами, входящими в состав пулов майнинга и зависящими от сервера пула, обеспечивающего функциональность полноценного узла. Функция майнинга на рис. 8.1 показана в виде черного круга с надписью «Майнер» или просто с буквой «М».

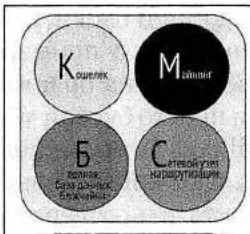
Пользовательские кошельки могут быть частью полноценного узла, обычно в том случае, когда клиент биткойна работает на настольном компьютере. Многие пользовательские кошельки, особенно работающие на мобильных устройствах (смартфонах и т. п.), являются SPV-узлами, и их количество постоянно увеличивается. Функция кошелька на рис. 8.1 показана в виде зеленого круга с надписью «Кошелек» или просто с буквой «К».

Кроме основных типов узлов, соответствующих пиринговому протоколу биткойна, существуют серверы и узлы, работающие по другим протоколам, таким как специальные протоколы пула майнинга и упрощенные протоколы доступа клиентов.

На рис. 8.2 показаны типы узлов, наиболее часто встречающиеся в расширенной биткойн-сети.

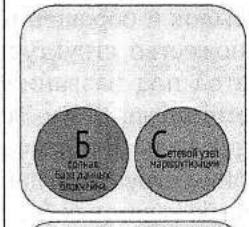
Расширенная биткойн-сеть

Основная биткойн-сеть, работающая по пиринговому протоколу биткойна, насчитывает от 5000 до 8000 активных («слушающих») узлов под управлением различных версий эталонной реализации биткойн-клиента (Bitcoin Core) и несколько сотен узлов, использующих другие реализации пирингового протокола биткойна, такие как Bitcoin Classic, Bitcoin Unlimited, BitcoinJ, Libbitcoin,



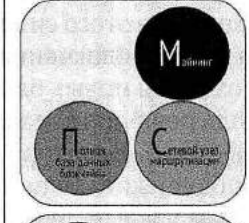
Эталонный клиент (Bitcoin Core)

Содержит кошелек, модуль майнинга, полную базу данных блокчейна и ПО маршрутизации в пиринговой биткойн-сети



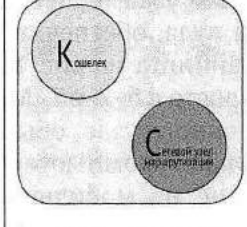
Полноценный узел блокчейна

Содержит полную базу данных блокчейна и ПО маршрутизации в пиринговой биткойн-сети



Один майнер

Содержит модуль майнинга с полной базой данных блокчейна и ПО маршрутизации в пиринговой биткойн-сети

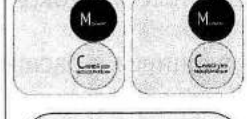


Упрощенный (SPV) кошелек

Содержит кошелек и ПО поддержки протокола пиринговой биткойн-сети без базы данных блокчейна

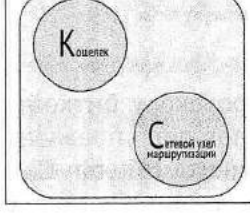
Серверы пула протоколов

Шлюзы-маршрутизаторы, соединяющие пиринговую биткойн-сеть с узлами, работающими по другим протоколам, например с узлами пула майнинга или Stratum-узлами



Узлы майнинга

Содержат модуль майнинга без базы данных блокчейна с ПО узла, поддерживающего протокол Stratum (S) или другие протоколы майнинга (П)



Упрощенный (SPV) кошелек Stratum

Содержит кошелек и ПО поддержки сетевого протокола Stratum без базы данных блокчейна

Рис. 8.2 ❖ Различные типы узлов в расширенной биткойн-сети

btcd и bcoin. Небольшой процент узлов в пиринговой биткойн-сети является также узлами майнинга, конкурирующими в процессе майнинга, при проверке корректности транзакций и в процессе создания новых блоков. Различные крупные компании взаимодействуют с биткойн-сетью через собственные полноценные узлы-клиенты на основе Bitcoin Core с поддержкой полной копии базы данных блокчейна и сетевой маршрутизации, но без функций майнинга и кошелька. Такие узлы работают как внешние (граничные) маршрутизаторы, обеспечивая функциональность других сервисов (сервисов обмена, кошельков, проводников блокчейна, обработку торговых платежных операций), созданных на их основе.

Расширенная биткойн-сеть (extended bitcoin network) включает сеть, работающую по пиринговому протоколу биткойна, описанному выше, а также узлы, работающие по специализированным протоколам. К основной пиринговой биткойн-сети подключено некоторое количество серверов пулов и шлюзов, которые обеспечивают соединения с узлами, работающими по другим протоколам. Узлами с альтернативными протоколами в основном являются узлы пула майнинга (см. главу 10) и упрощенные клиенты-кошельки, которые не содержат полной копии базы данных блокчейна.

На рис. 8.3 показана схема расширенной биткойн-сети с различными типами узлов, серверами-шлюзами, граничными маршрутизаторами и клиентами-кошельками, а также с различными протоколами, используемыми для их соединения друг с другом.

Сеть Bitcoin Relay Network

Несмотря на то что пиринговая биткойн-сеть удовлетворяет все основные потребности большинства типов узлов, в ней весьма часто возникают задержки, критичные для специализированных узлов майнинга биткойна.

Майнеры биткойнов участвуют в состязании с жесткими временны́ми ограничениями, целями которого являются решение задачи доказательства выполнения работы (Proof-of-Work) и расширение структуры данных блокчейна (см. главу 10). Включаясь в это состязание, майнеры биткойнов должны свести к минимуму время между распространением блока-победителя и началом следующего раунда состязания. В процессе майнинга сетевые задержки напрямую связаны с ограничениями на получение прибыли.

Bitcoin Relay Network – это сеть, которая пытается минимизировать задержки при передаче блоков между майнерами. Первоначальная версия сети Bitcoin Relay Network (http://www.bitcoinrelaynetwork.org) была создана основным разработчиком Мэтом Коральо (Matt Corallo) в 2015 году для обеспечения быстрой синхронизации блоков между майнерами с чрезвычайно малыми задержками. Сеть состояла из нескольких специальных узлов, размещенных в инфраструктуре Amazon Web Services по всему миру, и предназначалась для соединения большинства майнеров и пулов майнинга.

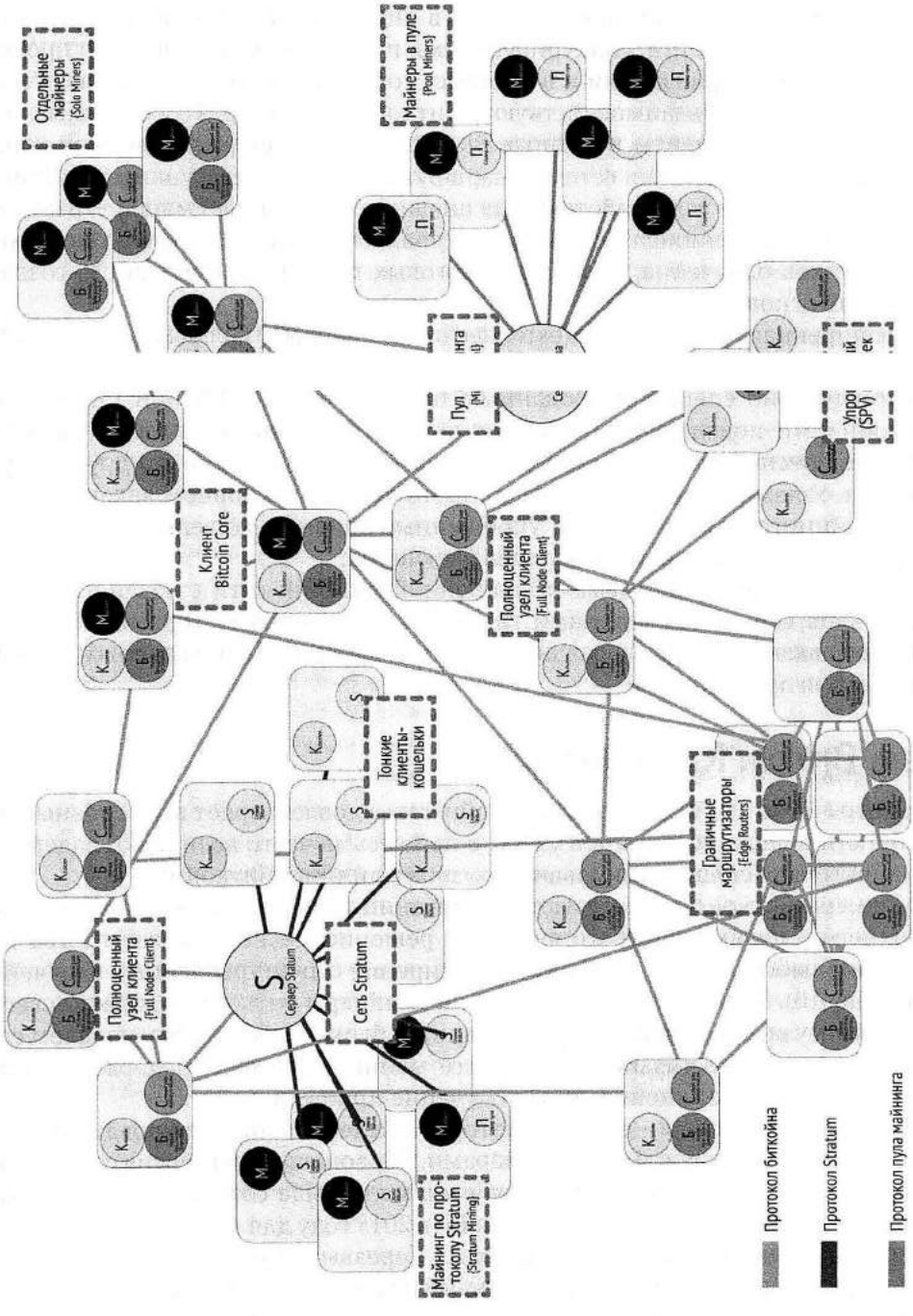


Рис. 8.3 ❖ Расширенная биткойн-сеть
с отображением различных типов узлов, шлюзов и протоколов

Первоначальная версия Bitcoin Relay Network была выведена из эксплуатации в 2016 году после появления новой версии Fast Internet Bitcoin Relay Engine, или FIBRE (http://bitcoinfibre.org), также созданной Мэтом Коральо. FIBRE – это ретрансляционная сеть на основе протокола UDP, которая передает блоки в сети специальных узлов. В сети FIBRE реализована оптимизация с целью создания компактных блоков (compact block) для уменьшения объема передаваемых данных и минимизации сетевых задержек.

Другой ретрансляционной сетью (пока находящейся в стадии предварительной разработки) является Falcon (http://www.falcon-net.org/about), основанная на исследованиях Корнеллского университета (Cornell University). Сеть Falcon использует сквозную, или транзитную, коммутацию пакетов (cut-through-routing) вместо буферизованной коммутации (store-and-forward) для уменьшения задержек посредством распространения отдельных фрагментов блоков сразу после их приема без ожидания получения всего блока полностью.

Ретрансляционные сети не заменяют пиринговую биткойн-сеть. Они представляют собой дополнительные сети, расширяющие возможности соединений между узлами со специальным предназначением. Скоростные автострады не заменяют второстепенные дороги, а лишь увеличивают пропускную способность между двумя пунктами с интенсивным движением, вот так же и в биткойн-системе необходимы «второстепенные дороги» для соединения с «автострадами».

Обследование биткойн-сети

Новый узел сразу после запуска обязательно должен обнаружить другие биткойн-узлы в сети, чтобы стать ее полноценным участником. Чтобы начать этот процесс, новый узел должен обнаружить, по меньшей мере, один существующий узел в сети и установить соединение с ним. Географическое расположение других узлов не имеет никакого значения, так как топология биткойн-сети определяется не по географическому принципу. Таким образом, любые существующие биткойн-узлы могут быть выбраны совершенно произвольно.

Для связи с известным партнером узлы устанавливают TCP-соединение обычно через порт 8333 (общеизвестный порт, один из используемых биткойн-системой) или через другой порт, если он предоставляется партнером. При установлении соединения узел начинает процедуру «рукопожатия» (handshake) (см. рис. 8.4), передавая сообщение `version`, которое содержит основную информацию для идентификации, включающую следующие данные:

- `nVersion` – версия протокола пиринговой биткойн-сети, на которой «говорит» клиент (например, 70002);
- `nLocalServices` – список локальных сервисов, поддерживаемых узлом, в настоящее время применяется только `NODE_NETWORK`;
- `nTime` – текущее время;
- `addrYou` – IP-адрес удаленного узла, как его «видит» узел-отправитель;
- `addrMe` – IP-адрес локального узла, определенный самим этим узлом;

- subver – дополнительный номер версии, обозначающий тип ПО, работающего на узле-отправителе (например, /Satoshi:0.9.2.1/);
- BestHeight – высота блока в структуре данных блокчейна на этом узле.

(Пример сетевого сообщения version см. на GitHub https://github.com/bitcoin/bitcoin/blob/d3cb2b8acfce36d359262b4afd7e7235eff106b0/src/net.cpp#L562.)

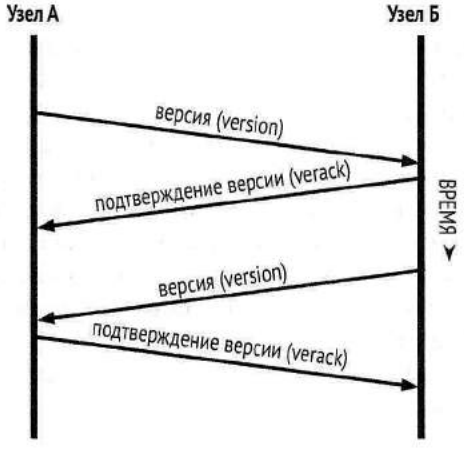


Рис. 8.4 ❖ Начальная процедура «рукопожатия» между партнерами

Сообщение version всегда является самым первым сообщением, посылаемым любому партнеру. Локальный партнер, получающий сообщение version, проверяет указанный удаленным партнером номер версии nVersion и решает, является ли ПО удаленного партнера совместимым. Если ПО удаленного партнера признано совместимым, то локальный партнер подтверждает прием сообщения version и устанавливает соединение, посылая сообщение verack.

Как ищет партнеров новый узел? Первый метод – запрос к сервису DNS с использованием нескольких «DNS-источников» («DNS seeds»), то есть серверов DNS, предоставляющих список IP-адресов биткойн-узлов. Некоторые из этих DNS-источников предлагают статический список IP-адресов стабильных биткойн-узлов, постоянно находящихся в режиме прослушивания. Некоторые DNS-источники являются специализированными реализациями протокола BIND (Berkeley Internet Name Daemon), которые возвращают случайно выбранное подмножество списка биткойн-адресов, собранного поисковым механизмом или очень долго существующим биткойн-узлом. Клиент Bitcoin Core содержит имена пяти различных DNS-источников (DNS-seeds). Разные владельцы этих DNS-источников и различия в их реализации определяют высокую степень надежности для первоначального процесса загрузки и раскрутки узла. В ПО клиента Bitcoin Core возможность обращения к DNS-источникам управляется ключом -dnsseed (по умолчанию установлено значение 1, разрешающее использование DNS-источников).

При использовании другого метода загружаемому узлу, который ничего не знает о сети, непременно должен быть передан IP-адрес как минимум одного биткойн-узла, после чего новый узел сможет устанавливать и другие соединения в дальнейшем. Аргументом командной строки `-seednode` можно воспользоваться для соединения с одним узлом только при первом появлении в сети, используя указанный узел как источник. После того как с помощью начального узла-источника новый узел «вошел» в сеть, клиент отключается от этого источника и обращается к новым обнаруженным партнерам.

Сразу после установления одного или нескольких соединений новый узел посылает соседям сообщение `addr`, содержащее его собственный IP-адрес. В свою очередь, соседи перенаправляют это сообщение `addr` своим соседям, тем самым гарантируя, что новый подключенный узел становится известным всем и ему будут обеспечены новые соединения, повышающие надежность. Кроме того, новый подключенный узел может отправить сообщение `getaddr` своим соседям, запрашивая список IP-адресов других партнеров. Таким образом, узел может находить партнеров для установления соединений и объявления о своем существовании всем прочим узлам сети. На рис. 8.5 показана схема процедуры определения адресов.

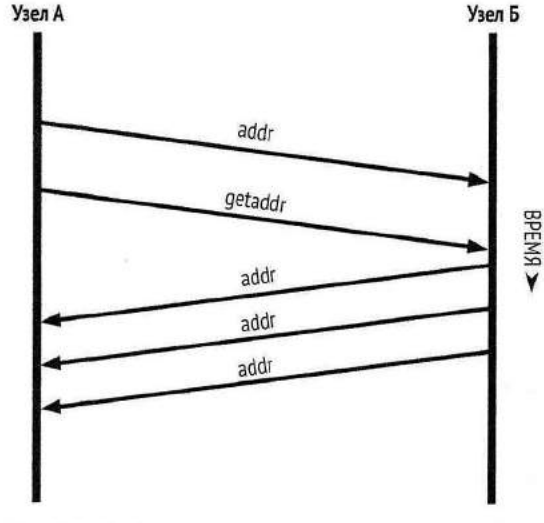


Рис. 8.5 ❖ Распространение собственного адреса и получение адресов новых партнеров

Узел обязательно должен установить соединения с несколькими партнерами, чтобы обеспечить наличие различных маршрутов в биткойн-сети. Маршруты не являются надежными – узлы появляются и исчезают, – поэтому узел должен продолжать поиск новых узлов-партнеров, чтобы сохранить свою работоспособность при потере старых соединений, а также чтобы помогать другим узлам

при их первоначальной загрузке. При первоначальной загрузке необходимо только одно соединение, потому что первый узел сразу может «представить» новичка своим партнерам, в свою очередь, они могут продолжить эту процедуру для других узлов сети. Также нет необходимости в излишних сетевых ресурсах для обеспечения соединений с относительно небольшим количеством узлов. После первоначальной загрузки новый узел запоминает самые свежие успешные соединения с партнерами, так что если случается перезагрузка, то он сможет быстро восстановить соединения с партнерами по сети. Если ни один из ранее установленных партнеров не отвечает на запрос соединения, то узел может снова воспользоваться узлами-источниками для повторного проведения процедуры первоначальной загрузки.

На узле, работающем под управлением ПО клиента Bitcoin Core, можно получить список соединений с партнерами с помощью команды `getpeerinfo`:

```
$ bitcoin-cli getpeerinfo
[
    {
        "addr" : "85.213.199.39:8333",
        "services" : "00000001",
        "lastsend" : 1405634126,
        "lastrecv" : 1405634127,
        "bytessent" : 23487651,
        "bytesrecv" : 138679099,
        "conntime" : 1405021768,
        "pingtime" : 0.00000000,
        "version" : 70002,
        "subver" : "/Satoshi:0.9.2.1/",
        "inbound" : false,
        "startingheight" : 310131,
        "banscore" : 0,
        "syncnode" : true
    },
    {
        "addr" : "58.23.244.20:8333",
        "services" : "00000001",
        "lastsend" : 1405634127,
        "lastrecv" : 1405634124,
        "bytessent" : 4460918,
        "bytesrecv" : 8903575,
        "conntime" : 1405559628,
        "pingtime" : 0.00000000,
        "version" : 70001,
        "subver" : "/Satoshi:0.8.6/",
        "inbound" : false,
        "startingheight" : 311074,
        "banscore" : 0,
        "syncnode" : false
    }
]
```

Чтобы отменить автоматическое управление списком партнеров и определить свой список IP-адресов, можно применить ключ `-connect=<IPAddress>` и задать один или несколько IP-адресов. При использовании этого ключа узел будет устанавливать соединение только с указанными IP-адресами вместо автоматического поиска и поддержки списка соединений с партнерами.

Если на соединении отсутствует трафик, то узлы периодически посылают сообщения для сохранения установленного соединения. Если узел не производит обмена данными по соединению в течение более 90 минут, то считается, что он отключился, и нужно начинать поиск нового партнера. Таким образом, сеть динамически устраняет периодически возникающие мелкие проблемы с отдельными узлами и сетью в целом и может вполне естественным образом расширяться и уменьшаться при необходимости без какого-либо централизованного управления.

Полноценные узлы

Полноценные узлы – это узлы, поддерживающие полную структуру данных блокчейна со всеми транзакциями. Возможно, более точно их следует называть «полноценные узлы блокчейна». На раннем этапе существования биткойн-системы все узлы были полноценными, в настоящее время клиент Bitcoin Core является полноценным узлом блокчейна. Тем не менее за последние два года были введены новые формы биткойн-клиента, которые не поддерживают полную структуру данных блокчейна, но работают как упрощенные клиенты. В следующем разделе мы рассмотрим их более подробно.

Полноценные узлы блокчейна поддерживают полную и актуальную копию структуры данных блокчейна биткойн-системы со всеми транзакциями, которые они независимо друг от друга создают и проверяют, начиная с самого первого блока (первичного блока – genesis block) и выстраивая цепочку до самого последнего известного в сети блока. Полноценный узел блокчейна может независимо и авторитетно проверить любую транзакцию, не обращаясь к ресурсам или какой-либо другой поддержке прочих узлов или источника информации. Полноценный узел блокчейна полагается только на сеть при получении обновленных данных о новых блоках транзакций, которые затем проверяются и включаются в локальную копию структуры данных блокчейна этого узла.

Поддержка функционирования полноценного узла блокчейна дает вам исключительный опыт работы с биткойн-системой: независимая проверка корректности всех транзакций без необходимости полагаться или доверять эти процедуры другим системам. Но не так-то просто обеспечить поддержку полноценного узла, потому что он требует более 20 Гб (дискового пространства) для постоянного хранилища полной структуры данных блокчейна. Вам потребуются жесткий диск весьма большого размера и два-три дня для синхронизации с сетью, и только после этого вы запустите полноценный узел. Это цена полной независимости и свободы от централизованного управления.

Существует несколько альтернативных реализаций полноценных узлов-клиентов блокчейна для биткойн-системы, созданных с применением различных языков программирования и программных архитектур. Но наиболее часто встречается эталонная реализация клиента Bitcoin Core, известная также под названием Satoshi client. Более 75% узлов в биткойн-сети работают под управлением различных версий Bitcoin Core. Они идентифицируются как «Satoshi» в строке подверсии, посылаемой в сообщении `version`, и отображаются в выводе команды `getpeerinfo`, описанной выше, например в таком виде: `/Satoshi:0.8.6/`.

Взаимная «инвентаризация»

Первое, что должен сделать полноценный узел после установления соединений с партнерами, – попытаться сформировать полную структуру данных блокчейна. Если это совершенно новый узел вообще без какой-либо базы данных блокчейна, то ему известен только один блок – первичный блок (genesis block), статически встроенный в программное обеспечение клиента. Начиная с блока #0 (с первичного блока), этот новый узел должен загрузить сотни тысяч блоков для синхронизации с сетью и восстановления полной структуры данных блокчейна.

Процесс синхронизации структуры блокчейна начинается с сообщения version, так как оно содержит поле `BestHeight`, текущую высоту цепочки блокчейна (количество блоков) узла-отправителя. Далее узел получит сообщения `version` от своих партнеров и узнает, сколько блоков содержится в их копиях структуры блокчейна. Узлы-партнеры обмениваются сообщением `getblocks`, содержащим хэш-значение (цифровой отпечаток) самого верхнего блока в локальной структуре блокчейна. Один из партнеров сможет определить полученное хэш-значение как принадлежащее не блоку, находящемуся на вершине цепочки, а более старому блоку, следовательно, можно сделать вывод: собственная локальная копия структуры блокчейна этого узла длиннее, чем копия его партнера.

Партнер, имеющий более длинную цепочку блокчейна, хранит большее количество блоков, чем другой узел, и может определить, какие блоки необходимы другому узлу, для того чтобы устранить это «отставание». Он идентифицирует первые 500 блоков, которыми нужно поделиться, и передает их хэш-значения в сообщении `inv` (inventory). Затем узел, не имеющий этих блоков, получает их, посылая последовательность сообщений `getdata` с запросами полных данных блоков и идентифицируя запрошенные блоки с помощью хэш-значений, ранее полученных в сообщении `inv`.

Например, предположим, что на узле имеется только первичный блок. Затем узел получает от партнеров сообщение `inv`, содержащее хэш-значения следующих 500 блоков в цепочке. Узел начинает запрашивать эти блоки у всех соединенных с ним партнеров, распределяя нагрузку и тем самым исключая перегруженность какого-то одного партнера огромным количеством запросов. Узел отслеживает количество блоков, находящихся «в стадии передачи» на каждом соединении, то есть блоков, запрошенных, но пока еще не принятых, и про-

веряет, не превышен ли лимит (MAX_BLOCKS_IN_TRANSIT_PER_PEER). Таким образом, если необходимо огромное количество блоков, узел будет запрашивать новую порцию только после завершения предыдущих запросов, позволяя партнерам управлять скоростью обновлений, чтобы не перегружать сеть. Каждый принятый блок добавляется в структуру данных блокчейна, как мы увидим в главе 9. Так постепенно создается локальная структура данных блокчейна, запрашиваются и принимаются новые блоки, и этот процесс продолжается до тех пор, пока узел не достигнет состояния, соответствующего состояниям всех прочих партнеров в сети.

Такой процесс сравнения локальной структуры данных блокчейна со структурами партнеров и получения всех отсутствующих блоков происходит каждый раз, когда узел возвращается в сеть после некоторого периода отключения. Не важно, находился ли узел в режиме офлайн лишь несколько минут и пропустил пару-тройку блоков или отсутствовал месяц и пропустил несколько тысяч блоков, работа всегда начинается с отправки сообщения getblocks, приема ответа inv и дальнейшей загрузки пропущенных блоков. На рис. 8.6 показана схема «инвентаризации» и работы протокола распространения блоков.

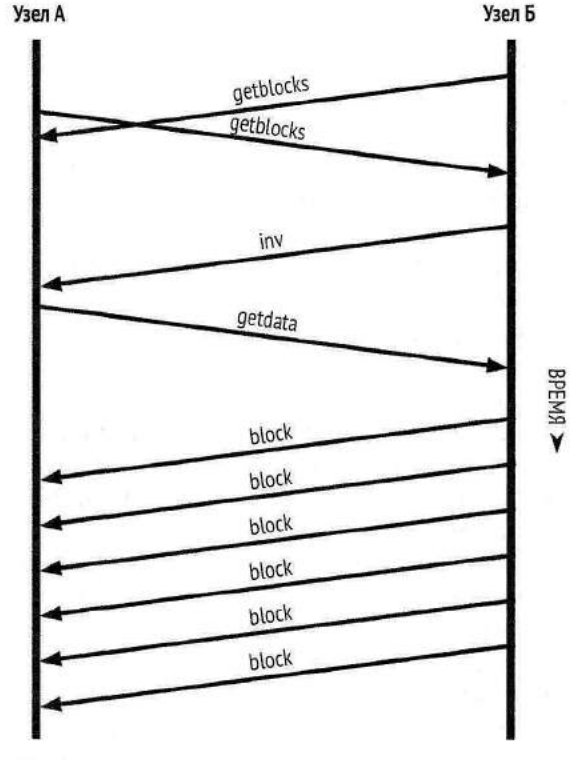


Рис. 8.6 ❖ Узел синхронизирует структуру данных блокчейна, получая недостающие блоки от партнера

Узлы с упрощенной проверкой платежей (SPV)

Не все узлы имеют возможность хранения полной структуры данных блокчейна. Многие биткойн-клиенты предназначены для работы на устройствах с ограниченными ресурсами и ограниченным сроком работы без подзарядки, таких как смартфоны, планшеты или встроенные системы. Для подобных устройств используется методика упрощенной проверки платежей (simplified payment verification – SPV), позволяющая им функционировать без хранения полной структуры данных блокчейна. Такие типы клиентов называются SPV-клиентами, или упрощенными клиентами (lightweight clients). По мере роста и распространения биткойн-сети SPV-узел становится наиболее часто встречающейся формой биткойн-узла, особенно для кошельков, хранящих биткойны.

SPV-узлы загружают только заголовки блоков, но не транзакции, включенные в каждый блок. Такая цепочка блоков без транзакций в 1000 раз меньше, чем полная структура данных блокчейна. SPV-узлы неспособны сформировать полную картину всех данных UTXO, доступных для расходования, поскольку таким узлам ничего не известно обо всех транзакциях в сети. SPV-узлы проверяют транзакции, используя несколько измененную методику, основанную на предоставлении партнерами копий необходимых фрагментов данных блокчейна по запросу.

В качестве аналогии представим себе полноценный узел как туриста в незнакомом городе, снабженного подробной картой, на которой указаны каждая улица и адрес каждого дома. По сравнению с ним, SPV-узел является туристом в незнакомом городе, который спрашивает дорогу у случайных встречных, чтобы постепенно определять направление движения, при этом ему известна только одна главная улица. Несмотря на то что оба туриста могут проверить существование любой улицы, посетив ее, турист без карты ничего не знает о расположении других улиц, более того, он даже не знает о существовании других улиц. Находясь перед домом с адресом Соборная, 23 (в оригинале – 23 Church Street), турист без карты не может узнать, существует ли в городе еще десяток адресов «Соборная, 23» или это именно тот адрес, который ему нужен. Для туриста без карты наилучшим решением является опрос достаточного количества людей с надеждой, что некоторые из них не будут пытаться подшутить над ним.

SPV-узел проверяет транзакции с точки зрения их глубины (depth) в цепочке блокчейна, а не с точки зрения их высоты (heght). Полноценный узел блокчейна формирует полностью проверенную (верифицированную) цепочку из тысяч блоков и транзакций, помещенных в структуру блокчейна (в прошлом), неизменно начиная с первичного блока, тогда как SPV-узел проверяет цепочку всех блоков (но не всех транзакций) и устанавливает связь этой цепочки с транзакцией, которая ему необходима.

Например, при проверке транзакции в блоке 300000 полноценный узел связывает все 300 000 блоков вниз по цепочке до первичного блока и формирует полную базу данных UTXO, устанавливая корректность и законность прове-

ряемой транзакции посредством подтверждения того, что соответствующие данные UTXO остаются неизрасходованными. SPV-узел не может проверить, остались ли неизрасходованными данные UTXO. Вместо этого SPV-узел устанавливает связь между проверяемой транзакцией и блоком, содержащим ее, используя путь в дереве Меркле (Merkle path) (см. раздел «Деревья Меркле» в главе 9). Затем SPV-узел ждет, когда поверх блока, содержащего проверяемую транзакцию, будут размещены шесть блоков с номерами с 300001 до 300006, после чего выполняет проверку, удостоверяющую глубину этого блока под блоками 300006–300001. Тот факт, что другие узлы в сети приняли блок 300000 и проделали необходимую работу по генерации еще шести новых блоков поверх проверяемого, является надежной гарантией того, что в проверяемой транзакции не было допущено двойное расходование.

SPV-узел невозможно убедить в том, что транзакция существует в блоке, если в действительности ее там нет. SPV-узел устанавливает факт существования транзакции в блоке, запрашивая путь в дереве Меркле в качестве доказательства и проверяя подлинность доказательства выполнения работы (Proof-of-Work) в цепочке блоков. И все же существование транзакции может быть «скрыто» от SPV-узла. SPV-узел может бесспорно доказать, что некоторая транзакция существует, но не способен проверить тот факт, что определенная транзакция, как, например, двойное расходование данных UTXO, не существует, потому что не располагает записями всех транзакций. Эта уязвимость может использоваться в атаке типа «отказ в обслуживании» (denial-of-service) или для атаки типа «двойное расходование» против SPV-узлов. Чтобы защититься, SPV-узел должен случайным образом устанавливать соединения с несколькими узлами, тем самым повышая вероятность того, что установлена связь, по крайней мере, с одним «честным» узлом. Такая необходимость в случайных соединениях означает, что SPV-узлы также уязвимы для атак типа «распадение сети» (network partitioning) или атак Сивиллы (Sybil attacks), при которых они устанавливают соединения с ложными узлами или ложными (под)сетями и не получают доступа к «честным» узлам реальной биткойн-сети.

Для большинства практических целей (задач) SPV-узлы с правильно установленными соединениями достаточно надежны с точки зрения безопасности, сохраняя баланс между потребностью в ресурсах, практичностью и безопасностью. Но в плане обеспечения самой надежной защиты ничто не может сравниться с полноценным узлом блокчейна.

Полноценный узел блокчейна верифицирует транзакцию, проверяя всю цепочку из тысяч блоков под ней, чтобы бесспорно доказать, что данные UTXO не израсходованы, тогда как SPV-узел просто проверяет, на какой глубине хранится соответствующий блок, исследуя несколько блоков, расположенных выше проверяемого.

Для получения заголовков блоков SPV-узлы пользуются сообщением `getheaders` вместо сообщения `getblocks`. Отвечающий партнер отправляет до 2000 заголовков блоков в одном сообщении `headers`. Этот процесс несколько отличается

от процесса получения полноценным узлом блоков со всем их содержимым. Кроме того, SPV-узлы устанавливают фильтр соединений с партнерами, чтобы отсеивать поток «будущих» блоков и транзакций, посылаемых партнерами. Все транзакции, представляющие интерес, извлекаются с помощью запроса `getdata`. Партнер генерирует сообщение `tx`, содержащее требуемые транзакции, в ответ на такой запрос. На рис. 8.7 показана схема синхронизации заголовков блоков.

Поскольку SPV-узлы вынуждены извлекать конкретные транзакции, чтобы избирательно проверить их, они создают еще и риск нарушения секретности (приватности). В отличие от полноценных узлов блокчейна, которые собирают все транзакции внутри каждого блока, запросы SPV-узлов на конкретные данные могут неумышленно открыть адреса соответствующих кошельков. Например, некоторая третья сторона, наблюдающая за сетью, может отслеживать все транзакции, запрашиваемые кошельком с SPV-узла, и использовать полученные данные для установления соответствия между биткойн-адресом и пользователем этого кошелька, тем самым нарушая право на защиту личной информации о пользователе.

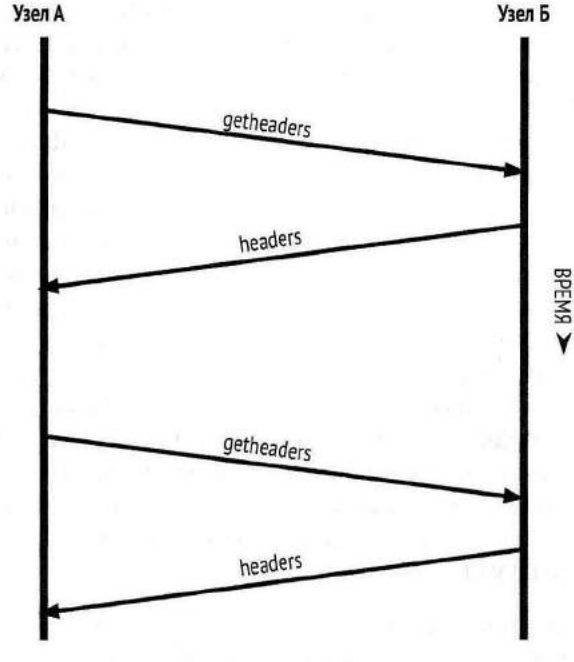


Рис. 8.7 ❖ SPV-узел,
выполняющий синхронизацию заголовков блоков

Вскоре после появления SPV-узлов (упрощенных узлов) разработчики биткойна добавили функциональную возможность под названием фильтр Блума (Bloom filter) для снижения риска нарушения приватности SPV-узлов. Фильтр

Блума позволяет SPV-узлам получать некоторое подмножество транзакций без точного указания требуемого конкретного адреса с помощью механизма фильтрации, который использует вероятностные методы вместо четко определенных шаблонов.

Фильтр Блума

Фильтр Блума (Bloom filter) – это фильтрующий механизм вероятностного поиска, способ описания требуемого шаблона (образца) без его точного определения. Фильтр Блума предлагает эффективный способ формирования шаблона поиска с обеспечением защиты приватности. Такие фильтры используются SPV-узлами при обращении к партнерам с запросами транзакций, соответствующих заданному шаблону, при этом точно не указывается, какие именно адреса, ключи или транзакции необходимо искать.

В предыдущем разделе в примере-аналогии из реальной жизни турист без карты выяснял маршруты, ведущие к адресу «Соборная, 23» («23 Church St.»). Спрашивая незнакомых людей о путях, ведущих к этой улице, турист невольно выдает свою цель. Фильтр Блума позволяет сформулировать вопрос так: «Есть ли поблизости улицы, названия которых заканчиваются буквами Р-Н-А-Я (в оригинале – R-C-H)?» Подобный вопрос открывает немного меньше информации о запрашиваемой цели, чем прямой вопрос о «Соборной, 23» («23 Church St.»). Применяя такую методику, турист может спрашивать о требуемом адресе с большими подробностями, например «заканчивается буквами О-Р-Н-А-Я» (в оригинале – U-R-C-H), или с меньшими подробностями: «заканчивается буквами А-Я» (в оригинале – H). Варьируя точность образца поиска, турист открывает больший или меньший объем информации, получая при этом более или менее точный результат соответственно. При вопросе с менее определенным образцом турист получит намного больше подходящих адресов и лучше обеспечит секретность, но многие результаты будут неверными. На вопрос с более определенным образцом турист получит меньше результатов, но секретность будет существенно снижена.

Фильтр Блума выполняет свою функцию, позволяя SPV-узлу определить образец поиска для транзакций с возможностью настройки сохранения большей точности или секретности. Более конкретно определенный фильтр Блума выдаст точные результаты, но платой за это будут открытые шаблоны, которыми интересуется этот SPV-узел, следовательно, и адреса, принадлежащие кошельку этого пользователя. Менее конкретно определенный фильтр Блума выдает больше данных о большем количестве транзакций, многие из которых не нужны этому узлу, но такой подход позволяет узлу сохранить большую степень секретности.

Как работает фильтр Блума

Фильтр Блума реализован как массив переменного размера, состоящий из N бинарных цифр (битовых полей) и переменного количества M хэш-функций.

Хэш-функции подбираются таким образом, чтобы всегда генерировать результат между 1 и N, соответствующий массиву бинарных цифр. Хэш-функции генерируются детерминированно, поэтому любой узел, применяющий фильтр Блума, всегда будет использовать одни и те же хэш-функции и получать одни и те же результаты для определенных входных данных. Фильтр Блума можно настраивать, выбирая различную длину (N) фильтра и различное количество (M) хэш-функций, тем самым изменяя уровень точности и соответствующий уровень секретности.

На рис. 8.8 показан пример очень маленького массива из 16 битов и набор из трех хэш-функций для наглядной демонстрации работы фильтра Блума.

Рис. 8.8 ❖ Пример упрощенного фильтра Блума с 16-битовым массивом и тремя хэш-функциями

Фильтр Блума инициализируется нулями во всех битовых полях. Для добавления шаблона в фильтр Блума этот шаблон хэшируется поочередно каждой хэш-функцией. Результатом применения первой хэш-функции к входным данным является число между 1 и N. Для соответствующего бита в массиве (проиндексированном от 1 до N) устанавливается значение 1, то есть записывается результат выполнения первой хэш-функции. Затем выполняется следующая хэш-функция и устанавливается соответствующий ее результату бит и т. д. После применения всех M хэш-функций шаблон поиска считается «записанным» в фильтре Блума как M битов, значения которых изменены с `0` на `1`.

На рис. 8.9 показан пример добавления шаблона «A» в простой фильтр Блума, изображенный на рис. 8.8.

Добавление второго шаблона выполняется очень просто, поскольку повторяется вышеописанный процесс. Шаблон хэшируется поочередно каждой хэш-функцией, результаты работы фиксируются с помощью установки для соответствующих битов значения `1`. Отметим, что поскольку фильтр Блума заполняется несколькими шаблонами, результат выполнения хэш-функции может соответствовать биту, для которого уже установлено значение `1`, и в этом

случае бит не изменяется. По существу, чем больше шаблонов записывается в перекрывающихся битовых полях, тем более насыщенным единичными битами становится фильтр, при этом точность фильтра снижается. Именно поэтому фильтр Блума является вероятностной структурой данных – он становится менее точным, если добавляется все больше шаблонов. Точность зависит от количества добавляемых шаблонов, сопоставляемого с размером битового массива (N), и числа хэш-функций (M). Битовый массив большего размера и большее количество хэш-функций позволяют записать больше шаблонов с высокой точностью. Битовый массив малого размера и небольшое количество хэш-функций позволяют записать меньше шаблонов и обеспечивают меньшую точность.

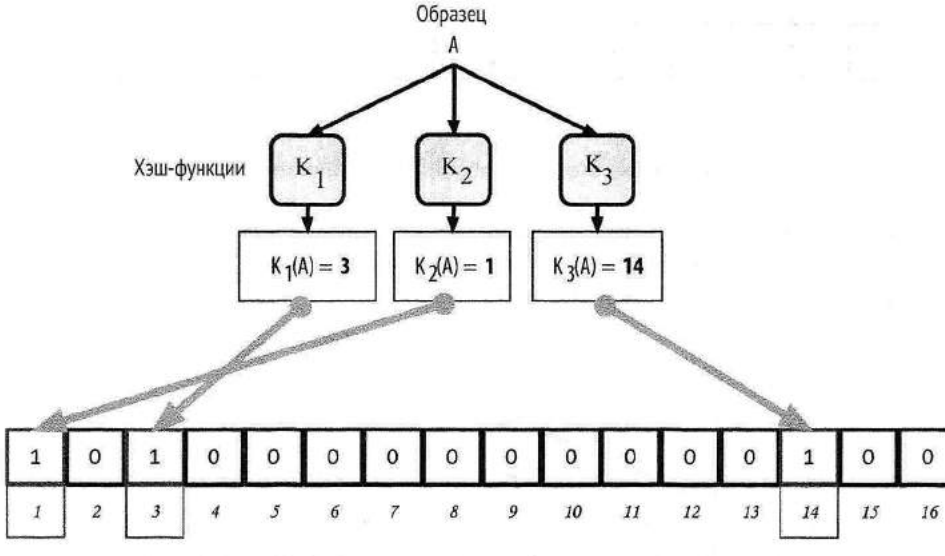


Рис. 8.9 ❖ Добавление шаблона «A» в простой фильтр Блума

На рис. 8.10 показан пример добавления второго шаблона «B» в простой фильтр Блума.

Чтобы проверить, является ли шаблон частью фильтра Блума, проверяемый шаблон хэшируется поочередно каждой хэш-функцией, и полученный в результате номер бита сравнивается с битовым массивом. Если все номера битов, полученные с помощью хэш-функций, соответствуют битам, установленным в 1, то этот шаблон, возможно (вероятно), записан в фильтре Блума. Так как биты могли быть установлены вследствие перекрытия нескольких шаблонов, ответ не может быть вполне определенным, это лишь вероятное допущение. Проще говоря, положительное совпадение проверяемого шаблона с фильтром Блума – это ответ «Вероятно, да».

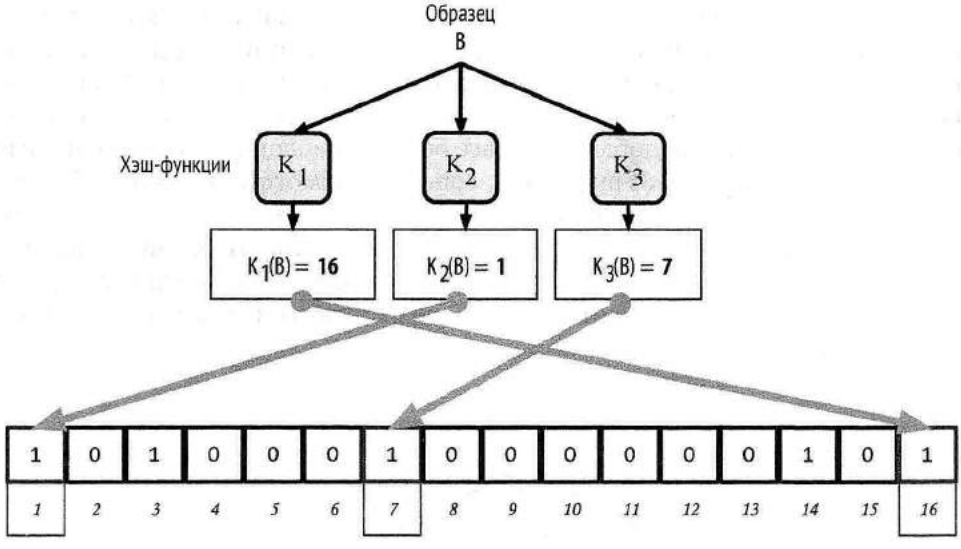


Рис. 8.10 ❖ Добавление второго шаблона «B» в простой фильтр Блума

На рис. 8.11 показан пример проверки наличия шаблона «X» в простом фильтре Блума. Соответствующие биты имеют значение 1, поэтому, вероятно, этот шаблон содержится в фильтре.

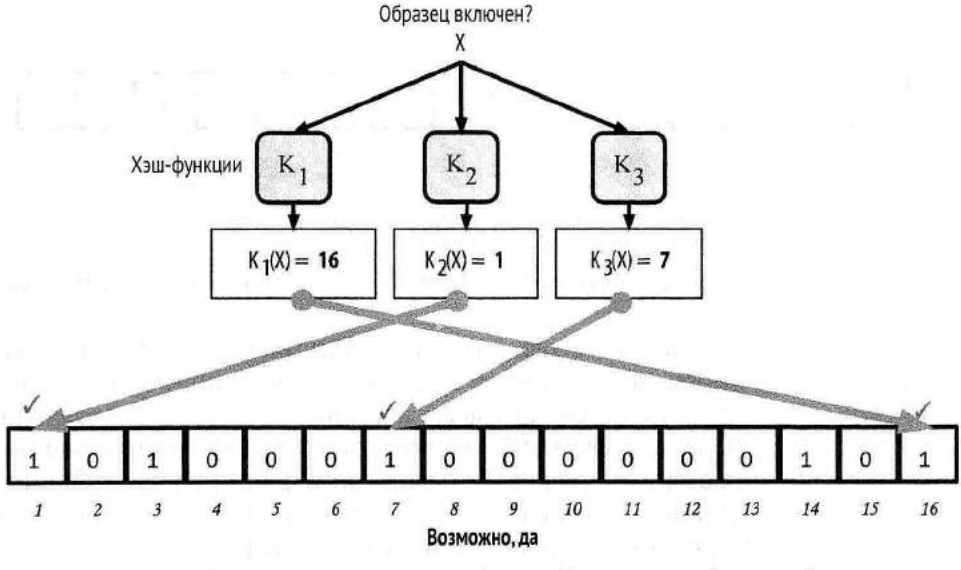


Рис. 8.11 ❖ Проверка наличия шаблона «X» в простом фильтре Блума. Результат – вероятностное позитивное соответствие, означающее «Вероятно (возможно)»

С другой стороны, если при проверке шаблона на наличие в фильтре Блума любой бит с вычисленным номером содержит значение `0`, это доказывает, что проверяемый шаблон не записан в фильтре Блума. Отрицательный результат не вероятностный, он вполне определенный. Проще говоря, отрицательный результат проверки – это ответ «Определенно нет».

На рис. 8.12 показан пример проверки наличия шаблона «Y» в простом фильтре Блума. Один из битов с вычисленными номерами имеет значение `0`, поэтому проверяемый шаблон определенно не содержится в фильтре.

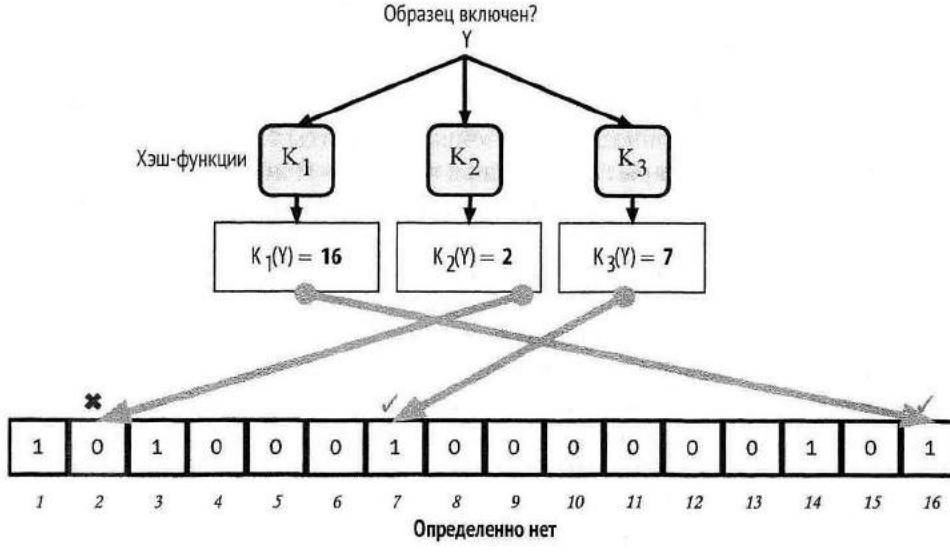


Рис. 8.12 ❖ Проверка наличия шаблона «Y» в простом фильтре Блума. Результат – явное отсутствие совпадения, означающее «Определенно нет»

Как SPV-узлы применяют фильтры Блума

Фильтры Блума используются для фильтрации транзакций (и содержащих их блоков), которые SPV-узел получает от партнеров, выбирая только интересующие его транзакции без раскрытия соответствующих адресов или ключей.

SPV-узел инициализирует фильтр Блума как «пустой», в этом состоянии фильтру не соответствуют никакие шаблоны. Затем SPV-узел формирует список всех адресов, ключей и хэш-значений, в которых он заинтересован. Это делается посредством извлечения хэш-значения открытого ключа, хэш-значения скрипта и идентификаторов транзакций из любых данных UTXO, управляемых кошельком этого узла. Затем SPV-узел добавляет каждый из выбранных объектов в фильтр Блума, так что «соответствие» фильтру будет достигнуто, если заданные шаблоны присутствуют в транзакции, при этом сами шаблоны не раскрываются.

После этого SPV-узел отправляет партнеру сообщение `filterload`, содержащее фильтр Блума, для использования на этом соединении. На стороне партнера фильтры Блума сверяются с каждой входящей транзакцией. Полноценный узел проверяет несколько частей транзакции на соответствие фильтру Блума, выполняя поиск совпадений в следующих элементах:

- идентификатор транзакции;
- компоненты данных из блокирующих скриптов каждого фрагмента выходных данных транзакции (каждый ключ и каждое хэш-значение в скрипте);
- каждый фрагмент входных данных транзакции;
- каждый компонент данных подписи входных фрагментов (или скриптов доказательств).

При проверке всех перечисленных элементов фильтры Блума можно использовать для поиска совпадений в хэшах открытых ключей, в скриптах, в значениях `OP_RETURN`, в открытых ключах в подписях или в любом компоненте смарт-контракта или сложного скрипта, в том числе и в тех компонентах, которые появятся в будущем.

После установки фильтра партнер проверяет с его помощью выходные данные каждой транзакции. Запрашивающему узлу отправляются только те транзакции, которые соответствуют фильтру.

В ответ на сообщение `getdata` от запрашивающего узла партнеры отправляют сообщение `merkleblock`, содержащее заголовки только тех блоков, которые соответствуют фильтру, и путь в дереве Меркле (см. раздел «Деревья Меркле» в главе 9) для каждой найденной транзакции. Затем партнеры начинают отправку сообщений `tx`, содержащих транзакции, соответствующие фильтру.

После начала передачи транзакций с полноценного узла на запрашивающий SPV-узел этот SPV-узел отбрасывает все ложные совпадения и использует транзакции с «правильными» совпадениями для обновления собственного набора данных UTXO и баланса кошелька. После обновления локального представления набора данных UTXO SPV-узел изменяет фильтр Блума, определяя в нем шаблоны для соответствия всем будущим транзакциям, ссылающимся на только что полученные данные UTXO. Затем полноценный узел использует новый фильтр Блума для поиска совпадений в новых транзакциях, и весь процесс повторяется.

Узел, устанавливающий фильтр Блума, может в интерактивном режиме добавлять шаблоны в этот фильтр, посылая сообщения `filteradd`. Для полной очистки фильтра Блума отправляется сообщение `filterclear`. Поскольку нет возможности удалять шаблоны из фильтра, узел должен полностью очистить фильтр, затем отправить новые шаблоны, если старый шаблон больше не нужен.

Сетевой протокол и механизм фильтра Блума для SPV-узлов определены и подробно описаны в документе BIP-37 (Peer Services) (https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki).

SPV-узлы и приватность

Узлы, реализующие механизм SPV, обладают более слабой защитой приватности, чем полноценный узел. Полноценный узел принимает все транзакции, поэтому не раскрывает информацию о том, использует ли он некоторый адрес в своем кошельке. SPV-узел получает отфильтрованный список транзакций, связанных с адресами в его кошельке. В результате снижается уровень защиты личных данных владельца.

Фильтры Блума представляют собой способ повышения уровня защиты личных данных и прочих секретов. Без фильтров SPV-узел был бы вынужден работать со списком конкретных адресов, интересующих его, при этом возникала бы серьезная уязвимость в системе защиты его личных данных. Но даже при использовании фильтров Блума мониторинг трафика SPV-клиента третьей стороной или прямое соединение с ним под видом узла пиринговой сети со временем позволяет собрать достаточный объем информации, чтобы узнать адреса в кошельке SPV-клиента.

Зашифрованные и защищенные соединения

Большинство новых пользователей биткойн-системы полагает, что сетевые соединения биткойн-узлов зашифрованы. В действительности эталонная реализация механизма сетевого обмена данными в биткойн-системе полностью открыта. Это не является существенным нарушением секретности и приватности для полноценных узлов, но для SPV-узлов представляет большую проблему.

В качестве меры защиты приватности и укрепления безопасности пиринговой биткойн-сети предлагаются два решения, обеспечивающие шифрование сетевых соединений: Tor Transport и P2P Authentication and Encryption (BIP-150/151).

Tor Transport

Tor – это аббревиатура от The Onion Routing network – названия программного проекта и сети, предлагающей шифрование и инкапсуляцию данных, передаваемых по случайно выбранным сетевым маршрутам. Такой подход обеспечивает анонимность, существенно затрудняет отслеживание и гарантирует соблюдение секретности (приватности).

Bitcoin Core предлагает несколько параметров конфигурации, которые позволяют запустить биткойн-узел в режиме передачи трафика через сеть Tor. Кроме того, Bitcoin Core предлагает скрытый сервис Tor, позволяющий другим Tor-узлам устанавливать соединение с вашим биткойн-узлом напрямую через сеть Tor.

В версии Bitcoin Core 0.12 узел автоматически начинает пользоваться скрытым сервисом Tor, если у него есть возможность установить соединение с локальным Tor-сервисом. Если ПО Tor установлено и процесс Bitcoin Core запу-

щен от имени пользователя, обладающего достаточными правами для доступа к куки (cookie) аутентификации Tor, то работа начинается автоматически. Используйте флаг debug для включения в Bitcoin Core режима отладки для сервиса Tor, например следующим образом:

```
$ bitcoind --daemon --debug=tor
```

В системных журналах (логах) вы должны увидеть запись «`tor: ADD_ONION successful`», означающую, что Bitcoin Core успешно добавил скрытый сервис для сети Tor.

Более подробные инструкции по запуску Bitcoin Core как скрытого сервиса Tor можно найти в документации по Bitcoin Core (*docs/tor.md*), а также в разнообразных онлайновых руководствах.

Аутентификация и шифрование в пиринговой сети

Два документа – BIP-150 и BIP-151 – добавляют поддержку P2P аутентификации и шифрования в пиринговой биткойн-сети. Эти два документа BIP определяют дополнительные сервисы, которые могут предлагаться совместимыми биткойн-узлами. Документ BIP-151 разрешает использование согласованного (по договоренности) шифрования для всех каналов обмена данными между двумя узлами, поддерживающими стандарт BIP-151. Документ BIP-150 предлагает дополнительную аутентификацию партнеров, позволяющую узлам проводить процедуру аутентификации для каждого постороннего объекта, используя механизм ECDSA и секретные ключи. Документ BIP-150 требует, чтобы перед процедурой аутентификации два узла обязательно установили зашифрованное соединение по стандарту BIP-151.

В январе 2017 года стандарты BIP-150 и BIP-151 еще не были реализованы в Bitcoin Core. Но эти два предложения реализованы, по крайней мере, в одном из альтернативных биткойн-клиентов под названием bcoin.

Документы BIP-150 и BIP-151 позволяют пользователям работать с SPV-клиентами, которые устанавливают соединение с доверенным полноценным узлом, используя шифрование и процедуру аутентификации для защиты приватности SPV-клиента.

Кроме того, аутентификация может использоваться для создания сетей из доверенных биткойн-узлов и для защиты от атак типа Man-in-the-Middle. Наконец, P2P-шифрование при широком распространении могло бы укрепить устойчивость биткойн-системы против попыток анализа трафика и слежки, нарушающей право на защиту личной информации, особенно в тоталитарных государствах, в которых использование Интернета жестко контролируется и отслеживается.

Стандарт определен в документах BIP-150 (Peer Authentication) (https://github.com/bitcoin/bips/blob/master/bip-0150.mediawiki) и BIP-151 (Peer-to-Peer Communication Encryption) (https://github.com/bitcoin/bips/blob/master/bip-0151.mediawiki).

Пулы транзакций

Почти каждый узел в биткойн-сети поддерживает временный список неподтвержденных транзакций, называемый пулом памяти (memory pool, mempool) или пулом транзакций (transaction pool). Узлы используют этот пул для наблюдения за транзакциями, которые известны в сети, но пока еще не включены в структуру данных блокчейна. Например, узел кошелька использует пул транзакций для отслеживания входящих платежей в кошелек пользователя, которые приняты в сети, но еще не подтверждены.

После того как транзакции приняты и проверены, они добавляются в пул транзакций и передаются соседним узлам для распространения по сети.

Некоторые реализации узлов также поддерживают отдельный пул транзакций-«сирот». Если входные данные транзакции ссылаются на транзакцию, которая еще неизвестна, например отсутствует «родительская» транзакция, то транзакция-«сирота» будет временно сохранена в пуле «сирот» до тех пор, пока не появится транзакция-предок.

При добавлении транзакции в основной пул проверяется пул «сирот» на наличие транзакций, ссылающихся на выходные данные новой транзакции (то есть потомков этой транзакции). Затем все подходящие транзакции-«сироты» проверяются на корректность и легальность. Если «сироты» корректны и легальны, то они удаляются из пула «сирот» и добавляются в пул транзакций, дополняя цепочку, которая начинается с родительской транзакции. Для каждой новой добавляемой транзакции, которая уже не является «сиротой», процесс рекурсивно повторяется, то есть проводится поиск всех ее последователей до тех пор, пока все последователи не будут найдены. В этом процессе прибытие родительской транзакции приводит к каскадному перестроению всей цепочки взаимозависимых транзакций путем воссоединения «сирот» с их родителями по всей цепочке.

Пул транзакций и пул «сирот» (если он реализован) хранятся в локальной оперативной памяти и не сохраняются на постоянных запоминающих устройствах, они динамически пополняются данными из приходящих сетевых сообщений. Сразу после запуска узла оба пула пусты, они постепенно заполняются новыми транзакциями, поступающими в сеть.

Некоторые реализации биткойн-клиента также поддерживают базу данных UTXO или пул UTXO, представляющий собой набор всех неизрасходованных транзакций в структуре данных блокчейна. Может показаться, что термин «пул данных UTXO» слишком похож на термин «пул транзакций», тем не менее он представляет отдельный набор данных. В отличие от пулов транзакций и «сирот», пул данных UTXO при инициализации не пуст, а содержит миллионы записей о неизрасходованных выходных данных транзакций по цепочке до самого первичного блока. Пул данных UTXO может храниться в локальной оперативной памяти или на устройстве внешней памяти в виде индексированной базы данных.

Пулы транзакций и «сирот» являются локальным представлением данных с точки зрения отдельного узла и могут существенно изменяться от узла к узлу в зависимости от того, когда узел начал свою работу или был перезагружен, тогда как пул данных UTXO представляет достигнутый (эмерджентный) консенсус в сети, следовательно, должны существовать очень малые различия между узлами. Более того, пулы транзакций и «сирот» содержат только неподтвержденные транзакции, в то время как пул данных UTXO содержит лишь подтвержденные выходные данные.

Глава 9

Блокчейн

Введение

Структура данных блокчейна (blockchain data structure) представляет собой упорядоченный список блоков транзакций с обратными ссылками (связями). Структура данных блокчейна может храниться как обычный «плоский» файл или в простой базе данных. Клиент Bitcoin Core хранит метаданные блокчейна, используя для этого базу данных LevelDB корпорации Google. Блоки связаны в обратном порядке, то есть каждый блок ссылается на предыдущий блок в цепочке. Визуально блокчейн часто представляют в виде вертикального стека, в котором блоки располагаются один поверх другого, а самый первый блок служит основанием стека. Подобная визуализация блоков, «укладываемых» один на другой, позволяет ввести термин «высота» (height) для обозначения расстояния от самого первого блока и термин «вершина» (top) или «верхушка» (tip) для обозначения блока, добавленного самым последним.

Каждый блок в структуре данных блокчейна идентифицируется по хэш-значению, сгенерированному с использованием алгоритма криптографического хэширования SHA256, примененного к заголовку блока. Кроме того, каждый блок ссылается на предыдущий блок, называемый родительским блоком (parent block), через поле хэш-значения предыдущего блока (previous block hash) в заголовке текущего блока. Другими словами, каждый блок содержит в собственном заголовке хэш-значение своего родителя. Такая последовательность хэш-значений, связывающая каждый блок с предыдущим, создает цепочку, неизменно ведущую обратно к блоку, который был создан самым первым, называемому первичным блоком (genesis block).

Любой блок имеет только одного родителя, но иногда временно может иметь несколько потомков. Каждый из таких потомков ссылается на один и тот же блок как на родительский и содержит одинаковое (родительское) хэш-значение в своем поле previous block hash. Несколько потомков порождается в процессе «разветвления» структуры блокчейна, временной ситуации, возникающей, когда различные блоки почти одновременно создаются разными майнерами (см. раздел «Разветвления структуры блокчейна» главы 10). В конце концов,

только один блок-потомок становится частью структуры блокчейна, и разветвление устраняется. Несмотря на то что любой блок может иногда иметь более одного потомка, каждый блок имеет одного и только одного родителя. Причина в том, что блок содержит только одно поле хэш-значения предыдущего блока, ссылающееся на единственного родителя.

Поле хэш-значения предыдущего блока располагается в заголовке блока, следовательно, влияет на хэш-значение текущего блока. При изменении идентификационных характеристик родителя изменяются и собственные идентификационные характеристики потомка. Если родитель каким-либо образом изменяется, то изменяется его (родительское) хэш-значение. Изменение хэш-значения родителя ведет к необходимости внесения соответствующего изменения в указателе (поле) previous block hash его потомка. В свою очередь, это изменяет хэш-значение потомка, что требует внесения изменений в поле-указатель потомка следующего поколения и изменяет его хэш-значение и т. д. Этот каскадный эффект гарантирует, что любой блок с многочисленными поколениями потомков не может быть изменен без обязательного повторного вычисления хэш-значений всех следующих за ним блоков. Поскольку такой повторный расчет потребует огромного объема вычислений (и соответствующего потребления энергии), существование длинной цепочки блоков делает полную хронологию структуры данных блокчейна практически неизменяемой, что является ключевым фактором для обеспечения безопасности биткойн-системы.

Одной из аналогий для структуры данных блокчейна являются слои в геологической формации пластов или слои в ядре ледника. Поверхностные слои могут изменяться по сезонам или даже полностью удаляться до того, как произойдет их окончательное осаждение. Но уже на глубине в несколько десятков сантиметров геологические слои становятся все более стабильными. На глубине в несколько сотен метров можно наблюдать картину далекого прошлого, которая оставалась неизменной в течение миллионов лет. В структуре данных блокчейна несколько самых свежих блоков может быть изменено в процессе перерасчета цепочки из-за ее разветвления. Шесть верхних блоков – это аналог нескольких десятков сантиметров верхнего слоя почвы. Но при более глубоком погружении в структуру блокчейна, ниже шести верхних блоков, более ранние блоки становятся все менее подверженными изменениям. На «глубине» 100 блоков обеспечивается такая степень стабильности, что появляется возможность расходования coinbase-транзакции, то есть транзакции, содержащей новый биткойн, созданный в процессе майнинга. Несколько тысяч новых блоков (месяц майнинга) и структура данных блокчейна представляют собой прочно обоснованную хронологию для любых практических целей. Несмотря на то что протокол всегда допускает замену цепочки более длинной цепочкой и разрешает отменять уже существующий блок, вероятность возникновения событий такого рода со временем снижается вплоть до бесконечно малой величины.

Структура блока

Блок – это контейнерная структура данных, объединяющая транзакции для включения их в общедоступный реестр – структуру данных блокчейна. Блок состоит из заголовка, содержащего метаданные, затем следует длинный список транзакций, главным образом определяющих общий размер блока. Длина заголовка блока равна 80 байтам, тогда как транзакция в среднем имеет длину 250 байтов, а в блок обычно включается не менее 500 транзакций. Таким образом, полная длина блока со всеми транзакциями приблизительно в 1000 раз больше длины заголовка. В табл. 9.1 описана структура блока.

Таблица 9.1. Структура блока

Размер	Поле	Описание
4 байта	Размер блока	Общий размер блока в байтах
80 байтов	Заголовок блока	Несколько полей, образующих заголовок блока
1–9 байтов (VarInt)	Счетчик транзакций	Количество транзакций в этом блоке
Переменный	Транзакции	Транзакции, записанные в этом блоке

Заголовок блока

Заголовок блока состоит из трех наборов метаданных блока. Первый набор – ссылка на хэш-значение предыдущего блока, соединяющего текущий блок с предыдущим блоком в структуре блокчейна. Второй набор метаданных – difficulty (сложность), timestamp (метка времени) и nonce – относится к процедуре майнинга и более подробно будет описан в главе 10. Третий набор метаданных – корень дерева Меркле, структуры данных, используемой для эффективного хранения всех транзакций в блоке. В табл. 9.2 описана структура заголовка блока.

Таблица 9.2. Структура заголовка блока

Размер	Поле	Описание
4 байта	Версия	Номер версии для отслеживания обновлений ПО/протокола
32 байта	Хэш предыдущего блока	Ссылка на хэш-значение предыдущего (родительского) блока в цепочке
32 байта	Корень дерева Меркле	Хэш-значение корня дерева Меркле, состоящего из транзакций этого блока
4 байта	Метка времени	Приблизительное время создания этого блока (в секундах по Unix-времени)
4 байта	Сложность целевой задачи	Сложность решения целевой задачи для этого блока по алгоритму Proof-of-Work
4 байта	Значение nonce	Случайное значение (счетчик), используемое в алгоритме Proof-of-Work

Значение nonce, сложность целевой задачи и метка времени используются в процессе майнинга и будут рассматриваться более подробно в главе 10.

Идентификаторы блока: хэш-значение заголовка блока и высота блока

Главным идентификатором блока является его криптографическое хэш-значение, цифровой отпечаток, вычисляемый посредством двукратного хэширования заголовка блока по алгоритму SHA256. Полученное в результате 32-байтовое хэш-значение называется хэш-значением блока (block hash), но более точным является термин «хэш-значение заголовка блока» (block header hash), так как для вычисления используется только заголовок. Например, шестнадцатеричное число `000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f` – это хэш-значение самого первого созданного блока в биткойн-системе. Хэш-значение однозначно и недвусмысленно идентифицирует блок и может быть независимо получено любым узлом с помощью простой операции хэширования заголовка блока.

Отметим, что хэш-значение блока в действительности не включается в структуру данных самого блока ни при передаче блока по сети, ни при сохранении блока в постоянном хранилище какого-либо узла как части структуры данных блокчейна. Вместо этого хэш-значение блока вычисляется каждым узлом при получении конкретного блока из сети. Хэш-значение блока может быть сохранено в отдельной таблице базы данных как часть метаданных блока для обеспечения индексирования и ускорения считывания блоков с дискового устройства.

Второй способ идентификации блока – его позиция в структуре блокчейна, называемая высотой блока (block height). Самый первый созданный блок расположен на высоте 0 (нуль). Это тот самый блок, который двумя абзацами выше был идентифицирован по своему хэш-значению `000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f`. Таким образом, любой блок идентифицируется двумя способами: по ссылке на хэш-значение блока или по ссылке на высоту блока. Каждый последующий блок, добавляемый «поверх» первого блока, размещается на одну позицию выше в структуре данных блокчейна, подобно коробкам, которые ставятся одна на другую. На 1 января 2017 года высота блока приблизительно равнялась 446 000, то есть около 446 000 блоков было размещено поверх первого блока, созданного в январе 2009 года.

В отличие от хэш-значения блока, высота блока не является однозначным идентификатором. Несмотря на то что один конкретный блок всегда будет иметь определенную и неизменяемую высоту, обратное высказывание не является истинным – высота блока не всегда однозначно определяет один конкретный блок. Два и более блоков могут располагаться на одной высоте, конкурируя за единственную позицию в структуре блокчейна. Такая ситуация подробно описана в разделе «Разветвления структуры блокчейна» главы 10. Высота блока также не является частью структуры данных блока, она вообще не хранится в блоке. Каждый узел динамически определяет позицию (высоту) блока в структуре блокчейна при получении конкретного блока из биткойн-

сети. Высота блока также может быть сохранена как метаданные в индексированной таблице базы данных для ускорения извлечения.

Хэш-значение блока (block hash) всегда однозначно и недвусмысленно идентифицирует один конкретный блок. Любому блоку также присуща определенная высота блока (block height). Но высота блока не всегда позволяет однозначно идентифицировать единственный блок. Два и более блоков могут конкурировать за единственную позицию в структуре данных блокчейна.

Первичный блок

Самый первый блок в структуре данных блокчейна называется первичным блоком (genesis block). Этот блок был создан в 2009 году и является общим «прародителем» всех блоков в структуре блокчейна: если начать с любого блока и следовать по цепочке в обратном хронологическом порядке, то в конечном итоге мы неизбежно приходим к первичному блоку.

Каждый узел всегда начинает работу со структурой блокчейна, состоящей как минимум из одного блока, потому что первичный блок статически закодирован в программном обеспечении биткойн-клиента, так что не может быть изменен. Каждый узел всегда «знает» хэш-значение и структуру первичного блока, фиксированное время его создания и даже единственную транзакцию, содержащуюся в этом блоке. Таким образом, каждый узел получает в свое распоряжение исходную позицию для структуры блокчейна, защищенный «корень», из которого создается надежная структура данных блокчейна.

Статически закодированный первичный блок можно подробно изучить в программном обеспечении клиента Bitcoin Core, в модуле *chainparams.cpp* (https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp).

Следующее идентифицирующее хэш-значение принадлежит первичному блоку:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

По этому хэш-значению вы можете выполнить поиск на любом сайте проводника блокчейна, например на blockchain.info, и непременно найдете страницу с описанием содержимого первичного блока с указанием URL, содержащего заданное хэш-значение:

- https://blockchain.info/block/000000000019d6689c085ae165831e934f763ae46a2a6c172b3f1b60a8ce26f;
- https://blockexplorer.com/block/000000000019d6689c085ae165831e934f763ae46a2a6c172b3f1b60a8ce26f.

Также можно воспользоваться эталонной реализацией клиента Bitcoin Core в командной строке:

```
$ bitcoin-cli getblock
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
{
```

```
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations" : 308321,
    "size" : 285,
    "height" : 0,
    "version" : 1,
    "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx" : [
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```


Первичный блок содержит скрытое сообщение. Входные данные coinbase-транзакции содержат текст: «The Times 03/Jan/2009 Chancellor on brink of second bail-out for banks». Это сообщение служит доказательством самой ранней даты создания этого блока, в точности воспроизводя заголовок британской газеты Times. Кроме того, эта фраза является ироническим напоминанием о важности независимой денежной системы именно в момент появления биткойнов и одновременного беспрецедентного всемирного финансового кризиса. Это сообщение включил в первичный блок Сатоши Накамото, создатель биткойнов.

Связывание блоков в структуру данных блокчейна

Полноценные биткойн-узлы поддерживают локальную копию структуры данных блокчейна, начинающуюся с первичного блока. Локальная копия структуры блокчейна постоянно обновляется при обнаружении новых блоков, используемых для наращивания цепочки. Когда узел получает новые блоки из сети, он проверяет их корректность, затем связывает их с существующей структурой блокчейна. Для установления связи узел исследует заголовок входящего блока в поисках поля хэш-значения предыдущего блока.

Предположим, например, что на узле имеются 277 314 блоков в локальной копии структуры блокчейна. Последним известным узлу является блок 277314 с хэш-значением заголовка:

```
00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249
```

Затем этот биткойн-узел получает из сети новый блок и выполняет его синтаксический разбор (парсинг) следующим образом:


```
{
    "size" : 43560,
    "version" : 2,
    "previousblockhash" :
```

```
        "0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
    "merkleroot" :
        "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
    "time" : 1388185038,
    "difficulty" : 1180923195.25802612,
    "nonce" : 4215469401,
    "tx" : [
        "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
#[... много транзакций пропущено ...]
        "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
    ]
}
```

Исследуя этот новый блок, узел находит поле `previousblockhash`, содержащее хэш-значение родительского блока. Найденное хэш-значение известно узлу – это хэш последнего блока в цепочке с высотой 277 314. Следовательно, полученный новый блок является потомком последнего блока в цепочке и расширяет существующую структуру блокчейна. Узел добавляет новый блок в конец цепочки, увеличивая высоту структуры блокчейна до 277 315. На рис. 9.1 показана цепочка из трех блоков, связанных с помощью ссылок в поле `previousblockhash`.

Деревья Меркле

Каждый блок в структуре данных блокчейна биткойн-системы содержит общую схему всех содержащихся в нем транзакций, используя для этого дерево Меркле (Merkle tree).

Дерево Меркле (Merkle tree), также называемое бинарным деревом хэш-значений (binary hash tree), – это структура данных, используемая для эффективного управления и проверки целостности крупных наборов данных. Деревья Меркле – это бинарные деревья, содержащие криптографические хэш-значения. Термин «дерево» (tree) используется в области ИТ для описания разветвленных структур данных, но такие деревья обычно изображаются растущими сверху вниз, с «корнем» в верхней части и листьями в нижней части диаграммы, как вы вскоре сами увидите в следующих примерах.

Деревья Меркле используются в биткойн-системе для объединения всех транзакций в блоке, для получения общего цифрового отпечатка всего набора транзакций в целом и для обеспечения весьма эффективного процесса проверки (верификации) действительности включения транзакции в блок. Дерево Меркле формируется из рекурсивно хэшируемых пар узлов, причем хэширование выполняется до тех пор, пока не будет получено одно хэш-значение, называемое корнем (root) или корнем дерева Меркле (Merkle root). Криптографический алгоритм хэширования, применяемый в деревьях Меркле биткойн-системы, – это алгоритм SHA256, применяемый дважды, также известный как double-SHA256.

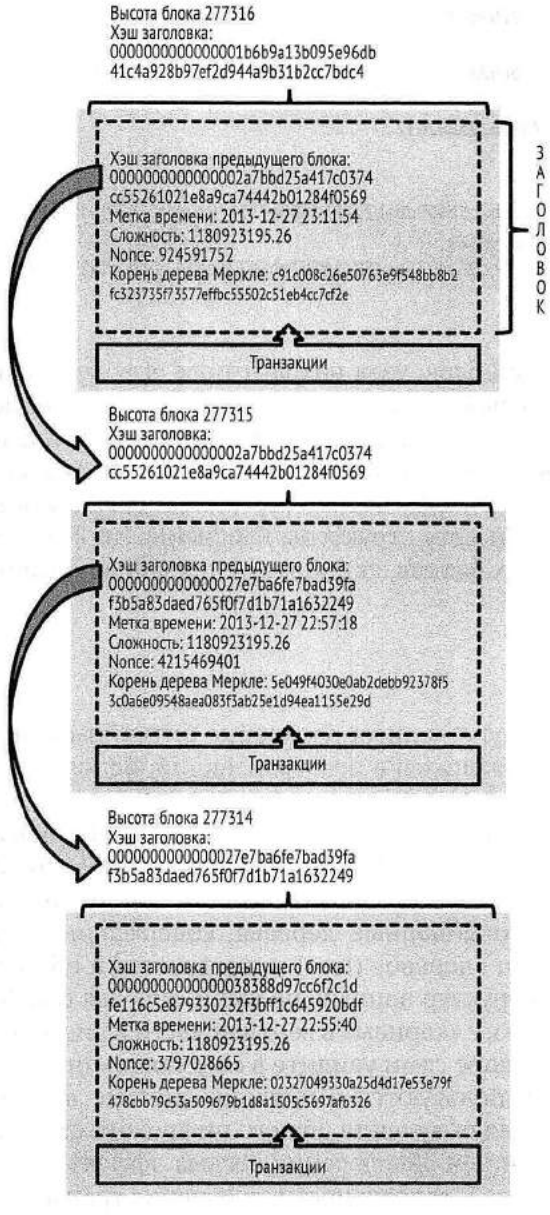


Рис. 9.1 ❖ Блоки, связанные в цепочку с помощью ссылок на хэш-значение заголовка предыдущего блока

При хэшировании и объединении в дерево Меркле N элементов данных для подтверждения факта включения любого элемента данных в дерево потребуется максимум $2*\log_2(N)$ вычислений, что делает эту структуру весьма эффективной.

Дерево Меркле формируется снизу вверх. В следующем примере мы начинаем с четырех транзакций A, B, C и D, представляющих листья (leaves) дерева Меркле, как показано на рис. 9.2. Транзакции не записываются в дерево Меркле, вместо этого их данные хэшируются, и полученные хэш-значения сохраняются в каждом соответствующем листе-узле как H_A, H_B, H_C и H_D:

```
H_A = SHA256(SHA256(Transaction A))
```

Затем пары листьев-узлов последовательно объединяются в родительский узел посредством сцепления двух соответствующих хэш-значений и хэширования полученного результата. Например, для создания родительского узла H_{AB} два 32-байтовых хэш-значения его потомков соединяются для формирования 64-байтовой строки. После этого полученная строка двукратно хэшируется для получения хэш-значения родительского узла:

```
H_AB = SHA256(SHA256(H_A + H_B))
```

Процесс продолжается до тех пор, пока не останется единственный узел в вершине, называемый корнем дерева Меркле. Это 32-байтовое хэш-значение записывается в заголовок блока и представляет собой общее итоговое хэш-значение для всех данных во всех четырех транзакциях. На рис. 9.2 показано, как формируется корень путем вычисления попарно объединяемых хэш-значений всех узлов.

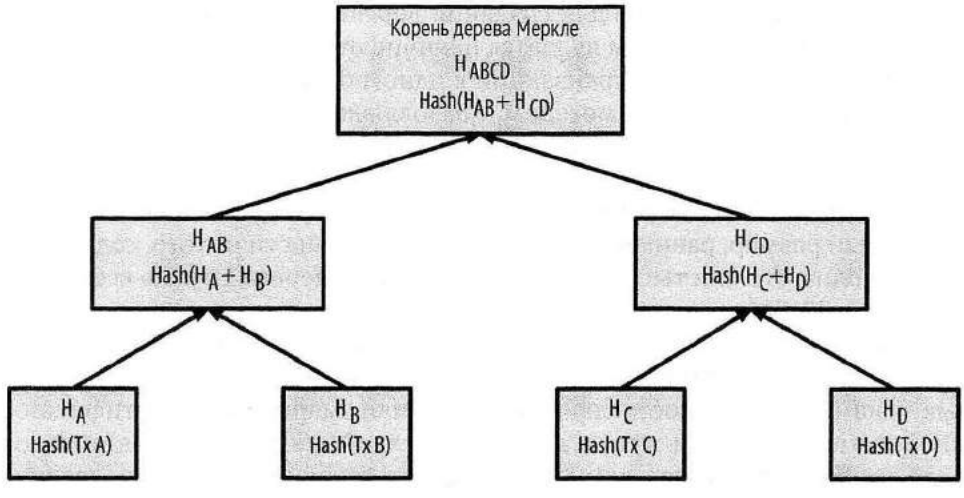


Рис. 9.2 ❖ Вычисление хэш-значений узлов в дереве Меркле

Поскольку дерево Меркле является бинарным деревом, в нем необходимо наличие четного числа листьев-узлов. Если число объединяемых транзакций нечетно, то хэш-значение последней транзакции дублируется для создания четного количества листьев-узлов. Такой подход называют сбалансированным

деревом (balanced tree). Эта методика показана на рис. 9.3, где дублируется хэш-значение транзакции C.

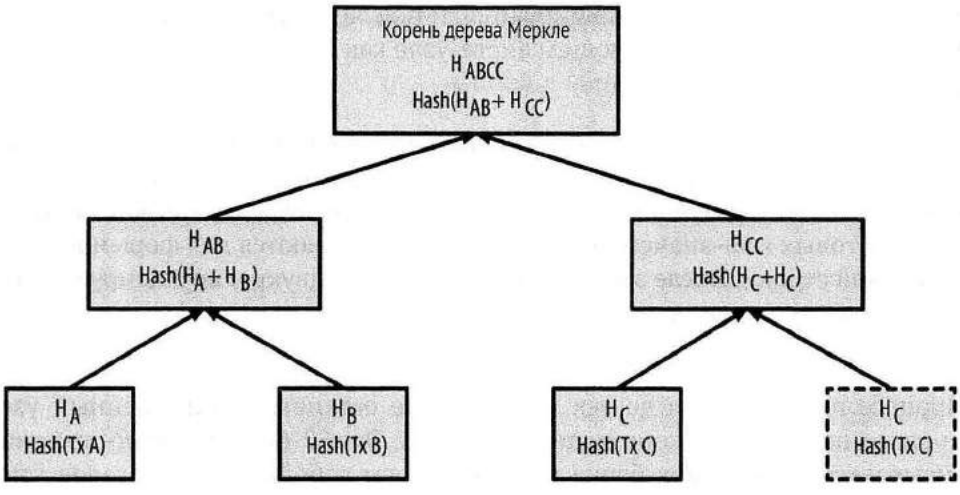


Рис. 9.3 ❖ Дублирование одного элемента данных, чтобы получить четное число элементов

Продемонстрированную выше методику формирования дерева из четырех транзакций можно обобщить для создания деревьев любого размера. В биткойн-системе обычным делом является наличие от нескольких сотен до тысячи (и даже немного больше) транзакций в одном блоке, при этом транзакции объединены точно таким же методом, позволяющим в итоге получить те же 32 байта данных как единственный корень дерева Меркле. На рис. 9.4 изображено дерево, сформированное из 16 транзакций. Отметим, что на схеме корень имеет больший размер, чем листья-узлы, но в действительности он имеет тот же самый размер, равный 32 байтам. Вне зависимости от того, содержится ли в блоке одна, сто или тысяча транзакций, корень дерева Меркле всегда объединяет их в 32-байтовое хэш-значение.

Для доказательства того, что некоторая транзакция действительно включена в блок, узел должен выполнить не более $\log_2(N)$ вычислений 32-байтовых хэш-значений, составляющих путь или маршрут аутентификации (authentication path) или путь в дереве Меркле (Merkle path), соединяющий проверяемую транзакцию с корнем дерева. Это особенно важно при увеличении количества транзакций, поскольку значение логарифма по основанию 2 от количества транзакций возрастает значительно медленнее. Это позволяет биткойн-узлам эффективно обрабатывать пути длиной в 10–12 хэш-значений (320–384 байта), которые могут обеспечить доказательство наличия одной транзакции в наборе из тысячи и более транзакций, размещенных в блоке размером в мегабайт.

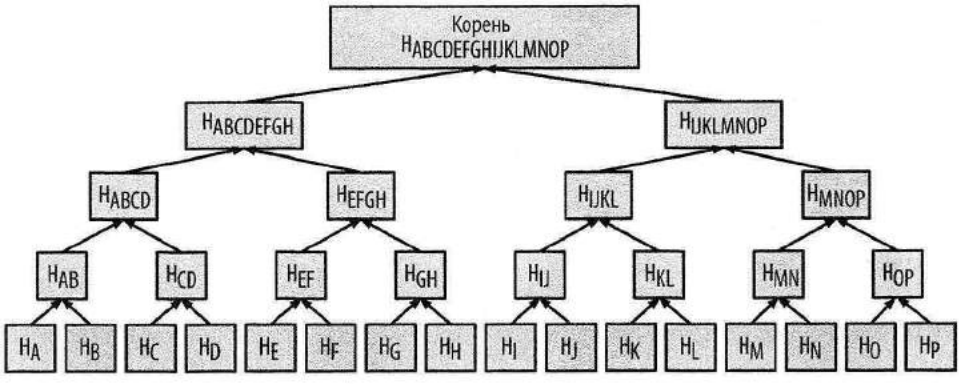


Рис. 9.4 ❖ Дерево Меркле,
объединяющее большое количество элементов данных

На рис. 9.5 узел может доказать, что транзакция K включена в блок, вычислив путь в дереве Меркле всего лишь из четырех 32-байтовых хэш-значений (общая длина 128 байтов). Путь состоит из этих четырех хэш-значений (показанных на рис 9.5 с закрашенным голубым фоном): H_L, H_{IJ}, H_{MNOP} и $H_{ABCDEFGH}$. С этими четырьмя хэш-значениями, предоставленными как путь аутентификации, любой узел сможет доказать, что значение H_K (с закрашенным черным фоном в нижней части схемы) включено в это дерево Меркле, вычислив попарно четыре дополнительных хэш-значения H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$ и корень дерева Меркле (на схеме они обозначены штриховыми контурными линиями).

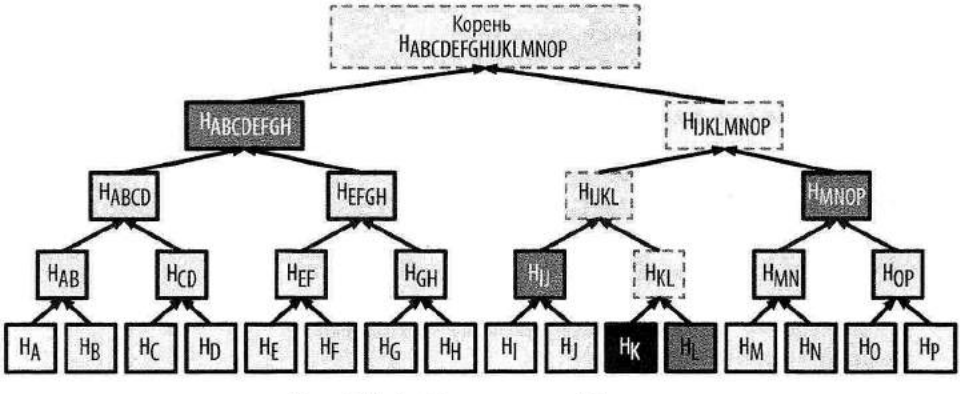


Рис. 9.5 ❖ Путь в дереве Меркле,
используемый для доказательства включения элемента данных в это дерево

Код в примере 9.1 демонстрирует процесс создания дерева Меркле из хэш-значений листьев-узлов с продвижением вверх до корня с использованием библиотеки libbitcoin, из которой взяты некоторые вспомогательные функции.

Пример 9.1 ❖ Создание дерева Меркле

```
#include <bitcoin/bitcoin.hpp>

bc::hash_digest create_merkle(bc::hash_list& merkle)
{
    // Остановить процесс, если список хэш-значений пуст
    if (merkle.empty())
        return bc::null_hash;
    else if (merkle.size() == 1)
        return merkle[0];

    // Если в списке более 1 хэш-значения, продолжить выполнение цикла...
    while (merkle.size() > 1)
    {
        // Если количество хэш-значений нечетно, продублировать последнее хэш-значение в списке
        if (merkle.size() % 2 != 0)
            merkle.push_back(merkle.back());
        // Теперь размер списка четный
        assert(merkle.size() % 2 == 0);

        // Новый список хэш-значений
        bc::hash_list new_merkle;
        // Цикл с обработкой 2 хэш-значений за одну итерацию
        for (auto it = merkle.begin(); it != merkle.end(); it += 2)
        {
            // Объединение (конкатенация) двух текущих хэш-значений
            bc::data_chunk concat_data(bc::hash_size * 2);
            auto concat = bc::make_serializer(concat_data.begin());
            concat.write_hash(*it);
            concat.write_hash(*(it + 1));
            assert(concat.iterator() == concat_data.end());
            // Хэширование объединенных хэш-значений
            bc::hash_digest new_root = bc::bitcoin_hash(concat_data);
            // Добавление результата в новый список
            new_merkle.push_back(new_root);
        }
        // Новый список хэш-значений
        merkle = new_merkle;

        // DEBUG -- вывод для отладки ------------------------------------
        std::cout << "Current merkle hash list:" << std::endl;
        for (const auto& hash: merkle)
            std::cout << " " << bc::encode_hex(hash) << std::endl;
        std::cout << std::endl;
        // ---------------------------------------------------------------
    }
    // Обработка списков заканчивается получением единственного элемента
    return merkle[0];
}

int main()
{
    // Замените эти хэш-значения реальными значениями из блока,
```

```
    // чтобы воспроизвести соответствующий корень дерева Меркле
    bc::hash_list tx_hashes{{
        bc::hash_lit
eral("0000000000000000000000000000000000000000000000000000000000000000"),
        bc::hash_lit
eral("0000000000000000000000000000000000000000000000000000000000000011"),
        bc::hash_lit
eral("0000000000000000000000000000000000000000000000000000000000000022"),
    }};
    const bc::hash_digest merkle_root = create_merkle(tx_hashes);
    std::cout << "Result: " << bc::encode_hex(merkle_root) << std::endl;
    return 0;
}
```

В примере 9.2 показан результат компиляции и запуска кода, создающего дерево Меркле.

Пример 9.2 ❖ Компиляция и выполнение кода примера, создающего дерево Меркле

```
$ # Компиляция кода примера merkle.cpp
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Запуск выполняемого файла merkle
$ ./merkle
Current merkle hash list:
  32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
  30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Current merkle hash list:
  d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

Эффективность деревьев Меркле становится очевидной при увеличении размеров структур. В табл. 9.3 показаны объемы данных, которые требуется обработать для доказательства того факта, что транзакция является частью рассматриваемого блока.

Таблица 9.3. Эффективность применения дерева Меркле

Количество транзакций	Прибл.размер блока	Длина пути (в хэшах)	Длина пути (в байтах)
16 транзакций	4 килобайта	4 хэш-значения	128 байтов
512 транзакций	128 килобайтов	9 хэш-значений	288 байтов
2048 транзакций	512 килобайтов	11 хэш-значений	352 байта
65 535 транзакций	16 мегабайтов	16 хэш-значений	512 байтов

Из таблицы видно, что при быстром увеличении размера блока от 4 Кб с 16 транзакциями до 16 Мб с 65 535 транзакциями путь в дереве Меркле, требуемый для доказательства наличия транзакции, увеличивается намного медленнее – от 128 байтов до 512 байтов. Используя деревья Меркле, узел может загрузить только заголовки блоков (по 80 байтов для каждого заголовка), сохраняя при этом возможность определения факта включения транзакции

в блок посредством получения небольшого по размеру пути в дереве Меркле от полноценного узла без необходимости хранения или передачи огромного объема данных блокчейна, который может достигать нескольких гигабайтов. Узлы, которые не поддерживают полную структуру данных блокчейна, называются узлами с упрощенной верификацией платежей (simplified payment verification, SPV), или просто SPV-узлами. Такие узлы используют пути в дереве Меркле для проверки транзакций без загрузки полного содержимого блоков.

Деревья Меркле и упрощенная верификация платежей (SPV)

Деревья Меркле широко используются SPV-узлами. SPV-узлы не имеют доступа ко всем транзакциям и загружают не блоки полностью, а только заголовки блоков. Чтобы проверить, действительно ли некоторая транзакция включена в блок без загрузки всех транзакций этого блока, используется путь аутентификации или путь в дереве Меркле.

Например, рассмотрим SPV-узел, интересующийся входящими платежами на адреса, содержащиеся в его кошельке. SPV-узел устанавливает фильтр Блума (см. раздел «Фильтры Блума» главы 8) на его соединениях с партнерами, чтобы ограничиться транзакциями, которые содержат только интересующие его адреса. Когда партнеры находят транзакцию, соответствующую фильтру Блума, они отправляют этот блок с помощью сообщения `merkleblock`. Сообщение `merkleblock` содержит заголовок блока и путь в дереве Меркле, который связывает требуемую транзакцию с корнем дерева Меркле в этом блоке. SPV-узел может использовать полученный путь в дереве Меркле для установления соединения с транзакцией в соответствующем блоке и проверки факта включения самой транзакции в блок. SPV-узел также использует заголовок блока для связи этого блока с остальной структурой блокчейна. Сочетание этих двух связей – между транзакцией и блоком и между блоком и структурой блокчейна – доказывает, что проверяемая транзакция действительно записана в структуре данных блокчейна. В совокупности SPV-узел принимает менее килобайта данных о заголовке блока и о пути в дереве Меркле. Этот объем данных в тысячу раз меньше, чем размер полного блока (в настоящее время около 1 Мб).

Тестовые структуры блокчейна в биткойн-системе

Возможно, вы удивитесь, узнав, что в биткойн-системе существует не одна структура данных блокчейна. «Главная» структура данных блокчейна для биткойна была создана Сатоши Накамото 3 января 2009 года. Она начинается с первичного блока, который мы рассматривали выше в этой главе, и называется mainnet. Но в биткойн-системе существуют и другие структуры блокчейна, используемые для тестирования: в настоящее время это testnet, segnet и regtest. Рассмотрим каждую из этих структур более подробно.

Testnet – «песочница» для тестирования биткойнов

Testnet – это название тестовой структуры данных блокчейна, сети и валюты, используемой для тестирования. Testnet представляет собой полнофункциональную действующую пиринговую сеть с кошельками, тестовыми биткойнами (testnet-койны), майнингом и всеми прочими функциональными возможностями главной сети mainnet. Имеются только два различия: testnet-койны не имеют никакой реальной ценности, а сложность майнинга должна быть достаточно низкой, чтобы упростить создание тестовых койнов (исключительно для тестирования).

Любая разработка программного обеспечения, предназначенного для реального использования в mainnet биткойна, должна сначала пройти тестирование в testnet на тест-койнах. Это защищает и разработчиков от потерь настоящих денежных средств из-за ошибок, и основную сеть от непредсказуемого поведения программ из-за тех же ошибок.

Но обеспечение бесполезности тестовой валюты и поддержка упрощенного майнинга являются непростой задачей. Игнорируя мнение разработчиков, некоторые люди используют мощное оборудование для майнинга (GPU и ASIC) в сети testnet. Это увеличивает сложность, делает невозможным майнинг с помощью обычных процессоров (CPU) и в итоге создает уровень сложности, достаточный для того, чтобы люди начали пользоваться тестовой валютой таким образом, как если бы она не была лишена ценности. В результате время от времени приходится останавливать работу testnet и перезапускать ее с новым первичным блоком, снижая уровень сложности.

Текущий вариант тестовой сети называется testnet3, то есть третья итерация testnet. Она была перезапущена в феврале 2011 года для снижения уровня сложности, завышенного в предыдущей итерации.

Следует помнить, что в testnet3 структура данных блокчейна имеет большой размер – в начале 2017 года он составлял более 20 Гб. Для синхронизации потребуются сутки и даже немного больше, а также полная занятость ресурсов вашего компьютера. Это не такой огромный объем, как в сети mainnet, но и не совсем малый. Одним из удобных способов запуска узла testnet является создание его в виртуальной машине (например, VirtualBox, Docker, Cloud Server и т. п.), выделенной специально для этой цели.

Использование testnet

Bitcoin Core, как и любое другое ПО биткойна, предоставляет полную поддержку операций в сети testnet вместо сети mainnet, а также позволяет выполнять майнинг тест-койнов и обеспечивает работу в качестве полноценного узла testnet.

Для запуска Bitcoin Core в сети testnet вместо mainnet необходимо воспользоваться ключом `-testnet` в командной строке:

```
$ bitcoind -testnet
```

В системных журналах (логах) вы увидите запись о том, что bicoind создает новую структуру блокчейна в подкаталоге *testnet3* каталога *bitcoind*, заданного по умолчанию:

```
bitcoind: Using data directory /home/username/.bitcoin/testnet3
```

Для установления соединения с bitcoind воспользуйтесь утилитой командной строки bitcoin-cli, но при ее использовании также необходимо переключиться в режим testnet:

```
$ bitcoin-cli -testnet getinfo
{
  "version": 130200,
  "protocolversion": 70015,
  "walletversion": 130000,
  "balance": 0.00000000,
  "blocks": 416,
  "timeoffset": 0,
  "connections": 3,
  "proxy": "",
  "difficulty": 1,
  "testnet": true,
  "keypoololdest": 1484801486,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "relayfee": 0.00001000,
  "errors": ""
}
```

Вы также можете воспользоваться командой getblockchaininfo для уточнения подробностей о структуре данных блокчейна в сети testnet3 и для наблюдения за ходом синхронизации:

```
$ bitcoin-cli -testnet getblockchaininfo
{
  "chain": "test",
  "blocks": 1088,
  "headers": 139999,
  "bestblockhash":
"0000000063d29909d475a1c4ba26da64b368e56cce5d925097bf3a2084370128",
  "difficulty": 1,
  "mediantime": 1337966158,
  "verificationprogress": 0.001644065914099759,
  "chainwork":
"0000000000000000000000000000000000000000000000000000044104410441",
  "pruned": false,
  "softforks": [

  [...]
```

Кроме того, можно работать в сети testnet3 с помощью других реализаций полноценного биткойн-узла, таких как btcd (написанного на языке Go) и bcoin (написанного на языке JavaScript), чтобы экспериментировать и более глубоко изучать языки программирования, инструментальные средства и программные среды.

В начале 2017 года сеть testnet3 поддерживала все функциональные возможности основной сети mainnet, а кроме того, функцию Segregated Witness (см. приложение Г), которая на тот момент еще не была введена в mainnet. Таким образом, сеть testnet3 можно использовать еще и для тестирования функции Segregated Witness.

Segnet – тестовая сеть для функции Segregated Witness

В 2016 году была введена в эксплуатацию тестовая сеть, специализированная для разработки и тестирования функции Segregated Witness (сокращенное название segwit, см. приложение Г). Эту тестовую структуру данных блокчейна назвали segnet. К ней можно подключиться с помощью специализированной версии (ветви) Bitcoin Core.

Но поскольку функция segwit была добавлена в тестовую сеть testnet3, отпала необходимость использования segnet для тестирования функциональных возможностей segwit.

Весьма вероятно, что в будущем мы увидим и другие тестовые структуры блокчейна, подобные segnet, специально ориентированные на тестирование отдельных функций или глобальных изменений архитектуры.

Regtest – локальная структура данных блокчейна

Regtest, или в полном варианте «Regression Testing», – это функция Bitcoin Core, позволяющая создавать локальную структуру данных блокчейна для тестирования. В отличие от testnet3, открытой и совместно используемой тестовой структуры данных блокчейна, структуры блокчейна regtest предназначены для запуска в виде закрытых систем для локального тестирования. Структура блокчейна regtest инициализируется «с нуля», создавая локальный первичный блок. Можно добавлять другие узлы в сеть или работать с единственным узлом для тестирования программного обеспечения Bitcoin Core.

Для запуска Bitcoin Core в режиме regtest используйте флаг `regtest`:

```
bitcoind: Using data directory /home/username/.bitcoin/regtest
```

Чтобы воспользоваться утилитой командной строки, также необходимо задать флаг `regtest`. Попробуем применить команду `getblockchaininfo` для исследования структуры блокчейна regtest:

```
$ bitcoin-cli -regtest getblockchaininfo
{
  "chain": "regtest",
  "blocks": 0,
  "headers": 0,
  "bestblockhash":
"0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206",
  "difficulty": 4.656542373906925e-10,
  "mediantime": 1296688602,
  "verificationprogress": 1,
  "chainwork":
"0000000000000000000000000000000000000000000000000000000000000002",
  "pruned": false,
  [...]
```

Мы видим, что в структуре пока еще нет блоков. Выполним майнинг нескольких (500) блоков и «заработаем» вознаграждение:

```
$ bitcoin-cli -regtest generate 500
[
  "7afed70259f22c2bf11e406cb12ed5c0657b6e16a6477a9f8b28e2046b5ba1ca",
  "1aca2f154a80a9863a9aac4c72047a6d3f385c4eec5441a4aafa6acaa1dada14",
  "4334ecf6fb022f30fbd764c3ee778fabbd53b4a4d1950eae8a91f1f5158ed2d1",
  "5f951d34065efeaf64e54e91d00b260294fcdfc7f05dbb5599aec84b957a7766",
  "43744b5e77c1dfece9d05ab5f0e6796ebe627303163547e69e27f55d0f2b9353",
   [...]
  "6c31585a48d4fc2b3fd25521f4515b18aefb59d0def82bd9c2185c4ecb754327"
]
```

Майнинг всех этих блоков займет лишь несколько секунд, что существенно упрощает тестирование. Если проверить баланс своего кошелька, то можно увидеть, что получено вознаграждение за первые 400 блоков (вознаграждение coinbase должно достигнуть глубины в 100 блоков, чтобы появилась возможность расходования заработанной суммы):

```
$ bitcoin-cli -regtest getbalance
12462.50000000
```

Использование тестовых структур блокчейна для разработки

Различные структуры блокчейна в биткойн-системе (`regtest`, `segnet`, `testnet3`, `mainnet`) предлагают разнообразные варианты сред тестирования для разработки ПО для биткойна. Используйте тестовые структуры блокчейна при разработке ПО для Bitcoin Core или альтернативного полноценного клиента консенсуса, а также при разработке таких приложений, как кошелек, сайт обмена валюты, сайт электронной коммерции, или даже при разработке инновационных смарт-контрактов и сложных скриптов.

Можно воспользоваться тестовыми структурами блокчейна для организации конвейерной разработки. Тестируйте свой код непосредственно в процессе разработки локально с использованием `regtest`. Когда программа будет готова для проверки в общедоступной сети, переходите в сеть `testnet`, чтобы проверить свой код в более динамической среде с быстро меняющимся ПО и набором приложений. Наконец, когда вы убедитесь, что ваша программа работает именно так, как предполагалось, переключайтесь в сеть `mainnet` для развертывания своего приложения в режиме реальной эксплуатации. При внесении изменений, улучшений и исправлении ошибок и т. п. снова запускайте конвейер разработки с самого начала, развертывая каждый измененный релиз ПО сначала в среде `regtest`, затем в `testnet` и, наконец, в среде реальной эксплуатации.

Глава 10

Майнинг и консенсус

Введение

Термин «майнинг» (mining) в некоторой степени вводит в заблуждение. По аналогии с добычей ценных полезных ископаемых он фокусирует наше внимание на вознаграждении за майнинг нового биткойна, создаваемого в каждом блоке. И хотя майнинг поощряется этим вознаграждением, его главной целью является вовсе не вознаграждение и не генерация новых биткойнов. Если рассматривать майнинг только как процесс создания новых денежных единиц, то вы ошибочно примете средства (стимулы) за основную цель этого процесса. Майнинг – это механизм, представляющий собой основу децентрализованной «расчетной палаты» (clearinghouse), с помощью которого проверяются, учитываются и расходуются все транзакции. Майнинг является изобретением, которое сделало биткойн особенной денежной единицей, это децентрализованный механизм защиты, основа пиринговой цифровой валюты.

Майнинг защищает биткойн-систему и обеспечивает достижение общего для всей сети консенсуса (consensus) без какого-либо центрального административного органа. Вознаграждение за «выпуск» новых денежных единиц и отчисления за транзакции представляют собой поощрительную схему, которая согласовывает действия майнеров с системой защиты сети, при этом одновременно осуществляется поддержка регулирования денежного обращения.

Создание новых биткойнов не является главной целью майнинга. Это поощрительная, стимулирующая система. Майнинг является механизмом, обеспечивающим децентрализацию средств и методов защиты в биткойн-системе.

Майнеры проверяют корректность и достоверность новых транзакций, после чего записывают их в глобальный реестр. Новый блок, содержащий транзакции, появившиеся после фиксации последнего блока, «майнится» в среднем каждые 10 минут, добавляя эти новые транзакции в структуру данных блокчейна. Транзакции, ставшие частью блока и добавленные в структуру блокчейна, считаются подтвержденными и принятыми (confirmed), что позволяет новым владельцам расходовать биткойны, полученные с помощью этих транзакций.

Майнеры принимают два типа вознаграждений за поддержку защиты, обеспечиваемой майнингом: новые денежные единицы, созданные каждым новым блоком, и отчисления (оплата) за все транзакции, включенные в сгенерированный блок. Для получения вознаграждения майнеры вступают в конкурентную борьбу, пытаясь решить сложную математическую задачу, основанную на алгоритме криптографического хэширования. Решение такой задачи, называемое доказательством выполнения работы (Proof-of-Work), включается в новый блок и представляет собой доказательство того, что майнер действительно затратил значительные вычислительные мощности на решение задачи. Конкуренция в процессе решения задачи по алгоритму доказательства выполнения работы для получения вознаграждения и за право записи транзакций в структуру данных блокчейна является основой модели защиты биткойн-системы.

Процесс назван майнингом, потому что вознаграждение (генерация новой денежной единицы) организовано по принципу, имитирующему принцип убывающей доходности (diminishing returns), действующий при добыче полезных ископаемых. Поддержка биткойнов как денежных единиц осуществляется через процесс майнинга, подобно тому, как центральный банк наращивает денежную массу, печатая новые банкноты. Максимальное количество вновь создаваемых биткойнов, которые майнер может добавить в блок, уменьшается приблизительно через каждые четыре года (или приблизительно через каждые 210 000 блоков). Вначале, в январе 2009 года, можно было включить 50 биткойнов в блок, а в ноябре 2012 года допустимое количество уменьшилось наполовину – 25 биткойнов в каждом новом блоке. В июле 2016 года лимит снова уменьшился вдвое и составил 12.5 биткойна на блок. По этой формуле вознаграждение за майнинг биткойнов уменьшается экспоненциально приблизительно до 2140 года, когда будут сгенерированы все возможные биткойны (20.99999998 миллиона). После 2140 года новые биткойны создаваться не будут.

Кроме того, майнеры биткойнов получают отчисления (оплату) за транзакции. В каждую транзакцию может быть включена ее оплата (transaction fee) в форме остатка или разности между суммами входных и выходных данных транзакции. Победивший в состязании за создание нового биткойна майнер «оставляет себе сдачу» из всех транзакций, включенных в блок-победитель. В настоящее время отчисления составляют 0.5% или немного меньше от общего дохода майнера, а основной доход майнер получает от генерации новых биткойнов. Тем не менее, поскольку вознаграждение со временем уменьшается, а количество транзакций в каждом блоке увеличивается, доля дохода от оплаты транзакций будет увеличиваться, по сравнению с долей от майнинга. Постепенно отчисления за транзакции будут становиться более весомыми, чем вознаграждение за майнинг, и станут более предпочтительным стимулом для майнеров. После 2140 года количество новых биткойнов в каждом блоке достигнет нулевой отметки, и процесс майнинга будет стимулироваться только отчислениями за транзакции.

В этой главе мы сначала рассмотрим майнинг как механизм поддержки финансового регулирования, затем перейдем к изучению более важной функции майнинга – механизма децентрализованного консенсуса, представляющего собой основу защиты биткойн-системы.

Для лучшего понимания процессов майнинга и достижения консенсуса мы снова обратимся к примеру с транзакцией Алисы, после того как она принята и добавлена в блок майнинговым оборудованием предпринимателя Цзина. Затем мы проследим процесс майнинга этого блока, добавление его в структуру данных блокчейна и принятие его всей биткойн-сетью с помощью процесса достижения консенсуса.

Экономика биткойна и создание валюты

Биткойны «печатаются» при создании каждого блока со строго определенной и постоянно уменьшающейся скоростью. Каждый новый блок, генерируемый в среднем с интервалом в 10 минут, содержит совершенно новые биткойны, созданные из ничего. Через каждые 210 000 блоков, или приблизительно через каждые четыре года, скорость выпуска денежных единиц уменьшается на 50%. В первые четыре года работы биткойн-сети каждый блок содержал 50 новых биткойнов.

В ноябре 2012 года скорость генерации новых биткойнов снизилась до 25 биткойнов на блок. В июле 2016 года произошло очередное сокращение до 12.5 биткойна на блок. Предел в 6.25 биткойна будет установлен в блоке номер 630000, майнинг которого будет выполнен в течение 2020 года. Таким образом, скорость выпуска новых денежных единиц экспоненциально уменьшается и составит 32 «сокращения наполовину» вплоть до блока номер 6720000 (его майнинг будет выполнен приблизительно в 2137 году), когда все эти сокращения приведут к минимальной денежной единице – 1 сатоши. Наконец, после генерации 6.93 миллиона блоков приблизительно в 2140 году будет выпущено около 2 099 999 997 690 000 сатоши, или почти 21 миллион биткойнов. После этого блоки не будут содержать новых биткойнов, а майнеры будут вознаграждаться исключительно за счет отчислений за транзакции. На рис. 10.1 показано общее количество биткойнов, поступающих в обращение с течением времени, при постоянном сокращении объема новых выпускаемых денежных единиц.

Максимальное количество полученных в результате майнинга денежных единиц представляет собой верхний предел (upper limit) возможных сумм вознаграждений за майнинг биткойнов. На практике майнер может выполнить майнинг блока, получая сумму, меньшую, чем полное вознаграждение. Такие блоки уже встречались в процессе майнинга и, вероятно, будут майниться и в будущем. Итогом этого станет уменьшение общего объема выпускаемых денежных единиц.

В коде примера 10.1 вычисляется общий объем (количество) биткойнов, которые будут сгенерированы в конечном итоге.

Рис. 10.1 ❖ График генерации биткойнов по временной шкале с учетом уменьшения выпускаемого объема по закону геометрической прогрессии

Пример 10.1 ❖ Скрипт, вычисляющий, сколько всего биткойнов будет сгенерировано

```
# Изначально вознаграждение за блок для майнеров составляло 50 BTC
start_block_reward = 50
# 210 000 блоков приблизительно каждые 4 года с 10-минутным интервалом создания нового блока
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 10.2 показан результат выполнения этого скрипта.

Пример 10.2 ❖ Выполнение скрипта max_money.py

```
$ python max_money.py
Total BTC to ever be created: 2099999997690000 Satoshis
```

Конечный и постоянно уменьшающийся объем выпуска денежных единиц обеспечивает надежную поддержку финансового регулирования, которое противостоит инфляции. В отличие от обычной валюты, печатаемой в неограниченном количестве центральным банком, биткойны никогда не будут подвержены инфляции из-за избыточного выпуска.

Дефляционная валюта

Наиболее важным и часто обсуждаемым последствием жестко определенного и постоянно уменьшающегося выпуска денежных единиц является тот факт,

что такая валюта по своей сущности стремится всегда оставаться дефляционной (deflationary). Дефляция (deflation) – это явление ревальвации денежных средств из-за несоответствия спроса и предложения, которое приводит к повышению ценности (и обменного курса) валюты. В противоположность инфляции дефляция означает, что покупательная способность денег возрастает со временем.

Многие экономисты высказывают мнение, что дефляционная экономика является бедствием, которого следует избегать любой ценой. Причина в том, что в период быстрой дефляции люди склонны накапливать деньги, а не тратить их, надеясь на падение цен в будущем. Подобное явление вскрылось во время «потерянного десятилетия» (Lost Decade) в Японии, когда крайне резкое падение спроса привело к неизбежному вводу валюты в дефляционную спираль.

Эксперты биткойна, в свою очередь, возражают, что сама по себе дефляция не так уж плоха. До некоторой степени дефляция ассоциируется с резким падением спроса, потому что это единственный пример дефляции, который мы рассмотрели. Для обычной валюты с возможностью неограниченного ее выпуска очень трудно создать вход в дефляционную спираль, если только не возникают полное обрушение спроса и нежелание (или неготовность) печатать новые деньги. Дефляция в биткойн-системе вызвана не падением спроса, а предварительно установленным и обусловленным ограничением.

Положительным аспектом дефляции является, разумеется, тот факт, что она противоположна инфляции. Инфляция приводит к медленному, но неизбежному снижению фактической ценности валюты, что в результате проявляется в форме скрытого налогообложения, бьющего по владельцам денежных сумм (вкладчикам, инвесторам и т. п.), чтобы «защитить» дебиторов (в том числе и самого крупного дебитора – правительство). Валюты, управляемые государством, теряют доверие из-за увеличения эмиссии денег, которую в дальнейшем можно будет компенсировать снижением их ценности за счет издержек вкладчиков.

Пока остается открытым вопрос: является ли дефляционная валюта проблемой, если она не управляется быстро меняющейся экономикой, или преимуществом, поскольку защищает от инфляции, а стратегия снижения выпуска денежных единиц значительно весомее, чем риски дефляции?

Децентрализованный консенсус

В предыдущей главе мы рассматривали структуру данных блокчейна, глобальный общедоступный реестр (список) всех транзакций, который каждый член биткойн-сети воспринимает как авторитетное свидетельство о праве владения собственностью.

Но каким образом члены сети приходят к единой, одинаковой для всех «истине», утверждающей, что кто-то владеет чем-то, без полного доверия друг другу? Все обычные платежные системы зависят от модели доверия, в которой центральное место занимает авторитетный орган управления, обеспечивающий функционирование службы клиринга платежей, проверяющей и узакони-

вающей все транзакции. В биткойн-системе нет такого центрального органа, но на каждом полноценном узле имеется полная копия общедоступного реестра, которому можно вполне доверять как достоверному регистрационному журналу. Структура блокчейна не создается каким-либо центральным органом управления, она формируется независимо каждым узлом сети. Каждый узел сети, действующий в соответствии с информацией, передаваемой по незащищенным сетевым соединениям, может тем или иным образом прийти к тому же выводу, что и его партнеры, и сформировать копию такого же общедоступного реестра, как и любой другой член сети. В этой главе рассматривается процесс достижения всеобщего консенсуса в биткойн-сети без какого-либо центрального управляющего органа.

Главным изобретением Сатоши Накамото является децентрализованный механизм достижения консенсуса (emergent consensus). Такой консенсус является эмерджентным (emergent), потому что не достигается в какой-либо явной форме – нет никаких голосований, не фиксируется конкретный момент достижения консенсуса. Консенсус представляет собой эмерджентный артефакт (сформированную сущность), полученный в результате асинхронного взаимодействия тысяч независимых узлов, соблюдающих простые правила. Все свойства и объекты биткойна, включая денежные единицы, транзакции, платежи и модель защиты, никоим образом не зависят от какого-либо центрального органа управления или доверительного попечения благодаря этому изобретению.

Децентрализованный консенсус в биткойн-системе формируется в результате взаимодействия четырех процессов, воспроизводимых независимо на всех узлах сети:

- независимая верификация (проверка) каждой транзакции каждым полноценным узлом на основе исчерпывающего списка критериев;
- независимое объединение этих транзакций в новые блоки узлами-майнерами, конкурирующими за предъявление фактических затрат на вычисления с помощью алгоритма доказательства выполнения работы (PoW);
- независимая верификация новых блоков каждым узлом сети и включение их в цепочку блоков;
- независимый выбор каждым узлом сети цепочки с максимальным суммарным объемом выполненных вычислений, предъявленных с помощью алгоритма доказательства выполнения работы (PoW).

В нескольких следующих разделах мы будем подробно рассматривать эти процессы и способы их взаимодействия для создания эмерджентных свойств общесетевого консенсуса, который позволяет любому биткойн-узлу формировать собственную копию достоверного, надежного (заслуживающего доверия), общедоступного глобального реестра.

Независимая верификация транзакций

В главе 6 мы наблюдали, как программное обеспечение кошелька создает транзакции, собирая необходимые данные UTXO, предоставляя соответствую-

щие разблокирующие скрипты, затем формируя новые выходные данные, связанные с новым владельцем. После этого созданная транзакция передается соседним узлам биткойн-сети, чтобы они могли распространить ее по всей сети.

Но прежде чем перенаправлять транзакции своим соседям, каждый биткойн-узел, принявший транзакцию, сначала выполняет ее верификацию (проверку). Проверка гарантирует, что в сети будут распространяться только корректные и законные транзакции, в то время как некорректные и незаконные транзакции отбрасываются первым же узлом, принявшим их.

Все узлы выполняют верификацию каждой транзакции, проверяя ее соответствие длинному списку следующих критериев:

- синтаксис транзакции и структура ее данных должны быть корректными;
- списки входных и выходных данных не должны быть пустыми;
- размер транзакции в байтах должен быть меньше значения параметра `MAX_BLOCK_SIZE`;
- значение каждого элемента выходных данных, а также общая сумма выходных данных должны находиться в диапазоне допустимых значений (меньше 21 миллиона биткойнов, но больше порогового значения dust);
- ни один из элементов входных значений не должен содержать параметров `hash=0, N=-1` (coinbase-транзакции не должны быть перенаправляемыми);
- параметр `nLocktime` должен быть равен `INT_MAX`, или же значения параметров `nLocktime` и `nSequence` должны соответствовать условиям `MedianTimePast`;
- размер транзакции в байтах должен быть больше или равен 100;
- количество операций подписи (SIGOPS), содержащихся в одной транзакции, должно быть меньше установленного лимита на операции подписи;
- разблокирующий скрипт (`scriptSig`) может только помещать числа в стек, а блокирующий скрипт (`scriptPubkey`) обязательно должен соответствовать стандартным формам `isStandard` (это позволяет отбрасывать «нестандартные» транзакции);
- обязательно должны существовать соответствующие друг другу транзакции – либо в пуле транзакций, либо в блоках основной ветви блокчейна;
- если в некоторой транзакции каждый фрагмент входных данных ссылается на выходные данные какой-либо другой транзакции в пуле, то такая транзакция непременно должна быть отброшена;
- для каждого фрагмента входных данных проводится поиск в основной ветви блокчейна и в пуле транзакций, чтобы найти выходные данные транзакции, на которые ссылаются входные данные. Если транзакция с соответствующими выходными данными не найдена для какого-либо элемента входных данных, то транзакция, содержащая этот фрагмент, будет считаться «сиротой». Она добавляется в пул транзакций-«сирот», если соответствующая транзакция уже не находится в этом пуле;

- если каждый фрагмент входных данных ссылается на выходные данные coinbase-транзакции, то транзакция, содержащая такие входные данные, должна иметь не менее `COINBASE_MATURITY` (100) подтверждений;
- каждый фрагмент входных данных непременно должен ссылаться на реально существующие выходные данные, которые не должны быть уже израсходованы;
- с использованием выходных данных транзакции, на которые ссылаются соответствующие входные данные, и извлечением последних проводится проверка каждого фрагмента входных данных, а также их общей суммы на нахождение в допустимом диапазоне значений (менее 21 миллиона биткойнов, но больше 0);
- транзакция отбрасывается, если сумма значений входных данных меньше суммы значений выходных данных;
- транзакция отбрасывается, если отчисление за транзакцию слишком мало (меньше `minRelayTxFee`) для передачи в пустой блок;
- разблокирующие скрипты для каждого фрагмента входных данных обязательно должны проверяться на соответствие блокирующим скриптам выходных данных.

Реализацию всех этих условий можно подробнейшим образом рассмотреть в функциях `AcceptToMemoryPool`, `CheckTransaction` и `CheckInputs` в программном коде Bitcoin Core. Отметим, что условия со временем изменяются, чтобы обеспечить защиту от новых типов DoS-атак, а иногда даже для некоторого смягчения правил, чтобы создать возможность включения новых типов транзакций.

С помощью независимой верификации каждой принятой транзакции перед ее дальнейшим продвижением в сети любой узел формирует пул корректных и легальных (но пока еще не подтвержденных) транзакций, который называют пулом транзакций (transaction pool), пулом памяти (memory pool) или просто mempool.

Узлы майнинга

Некоторые узлы биткойн-сети представляют собой особые узлы, называемые майнерами (miners). В главе 1 мы познакомились с Цзином, студентом, изучающим инженерию компьютерных систем в Шанхае (Китай), который является майнером биткойнов. Цзин зарабатывает биткойны, обеспечивая работу «комплекса или фермы майнинга» (mining rig), то есть специализированной компьютерной аппаратной системы, предназначенной исключительно для майнинга биткойнов. Специализированное оборудование майнинга Цзина соединено с сервером, на котором функционирует полноценный биткойн-узел. В отличие от Цзина, некоторые майнеры обходятся без полноценного узла, как мы увидим в разделе «Пулы майнинга» ниже в этой главе. Как и любой другой полноценный узел, узел Цзина принимает и распространяет неподтвержденные транзакции в биткойн-сети. Но, кроме того, узел Цзина объединяет эти транзакции в новые блоки.

Узел Цзина отслеживает («прослушивает») появление новых блоков, распространяемых в биткойн-сети точно так же, как это делают все узлы. Но прибытие нового блока имеет особое значение для узла майнинга. Конкуренция между майнерами фактически завершается при распространении нового блока, который действует как объявление победителя. Для майнеров получение нового корректного блока означает, что кто-то другой выиграл состязание, а они проиграли. Тем не менее конец одного раунда состязания также является началом следующего раунда. Новый блок представляет собой не просто клетчатый финишный флаг, сигнализирующий о завершении гонки, это одновременно и выстрел стартового пистолета, начинающий новую гонку за следующим блоком.

Объединение транзакций в блоки

После проверки корректности и достоверности транзакций биткойн-узел добавляет их в пул памяти (memory pool) или пул транзакций (transaction pool), где транзакции ожидают своей очереди на включение в блок (то есть в процесс майнинга). Узел Цзина собирает, проверяет и передает новые транзакции точно так же, как любой другой узел. Но, в отличие от прочих узлов, узел Цзина, кроме всего прочего, объединяет эти транзакции в блок-кандидат (candidate block).

Понаблюдаем за блоками, которые были созданы в то время, когда Алиса купила чашку кофе в кафе Боба (см. раздел «Покупка чашки кофе» в главе 1). Транзакция Алисы была включена в блок 277316. Для более наглядной демонстрации концепций в этой главе предположим, что майнинг этого блока был выполнен на системе Цзина, и проследим, как транзакция Алисы становится частью этого нового блока.

Узел майнинга Цзина поддерживает локальную копию структуры данных блокчейна. К моменту покупки Алисой чашки кофе узел Цзина сформировал цепочку до блока 277314. Узел Цзина отслеживает транзакции при попытке майнинга нового блока, а также наблюдает за блоками, создаваемыми другими узлами. В процессе майнинга узел Цзина получает блок 277315 из биткойн-сети. Прибытие этого блока означает завершение состязания за блок 277315 и начало нового состязания за создание блока 277316.

В течение предыдущих 10 минут, когда узел Цзина искал решение для блока 277315, он также собирал транзакции для включения их в очередной блок. К текущему моменту в пуле памяти собрано несколько сотен транзакций. При получении блока 277315 и проверке его корректности узел Цзина также проверяет все транзакции в своем пуле памяти и удаляет те, которые включены в блок 277315. Транзакции, остающиеся в пуле памяти, продолжают оставаться неподтвержденными и ожидают записи в новый блок.

Узел Цзина немедленно начинает создание нового пустого блока-кандидата для блока 277316. Такой блок называется блоком-кандидатом (candidate block),

потому что он пока еще не является корректным блоком, так как не содержит достоверного доказательства выполнения работы (PoW). Блок становится корректным, только если майнер найдет правильное решение по алгоритму доказательства выполнения работы (PoW).

После того как узел Цзина объединил все транзакции из пула памяти, новый блок-кандидат содержит 418 транзакций с общей суммой отчислений за транзакции, равной 0.09094928 биткойна. Этот блок в структуре данных блокчейна вы можете увидеть с помощью интерфейса командной строки клиента Bitcoin Core, как показано в примере 10.3.

Пример 10.3 ❖ Использование интерфейса командной строки для извлечения блока 277316

```
$ bitcoin-cli getblockhash 277316

000000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock 000000000000000001b6b9a13b095e96db41c4a928b97ef2d9\
44a9b31b2cc7bdc4

{
    "hash" : "000000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
    "confirmations" : 35561,
    "size" : 218629,
    "height" : 277316,
    "version" : 2,
    "merkleroot" :
"c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
    "tx" : [
        "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
        "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",

        ... еще 417 транзакций...

        ],
    "time" : 1388185914,
    "nonce" : 924591752,
    "bits" : "1903a30c",
    "difficulty" : 1180923195.25802612,
    "chainwork" :
"0000000000000000000000000000000000000000000000934695e92aaf53afa1a",
    "previousblockhash" :
"000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569"
}
```

Coinbase-транзакция

Самой первой транзакцией в любом блоке является особая транзакция, называемая coinbase-транзакцией (coinbase transaction). Эта транзакция создается узлом Цзина (в рассматриваемом примере) и содержит его вознаграждение (reward) за затраты на майнинг.

Во время майнинга блока 277316 вознаграждение составляло 25 биткойнов за каждый новый блок. После этого уже завершился очередной период «сокращения наполовину». В июле 2016 года вознаграждение за новый блок снизилось до 12.5 биткойна. Очередное сокращение наполовину снова произойдет при достижении высоты в 210 000 блоков в 2020 году.

Узел Цзина создает coinbase-транзакцию как платеж в собственный кошелек: «Выплатить по адресу Цзина 25.09094928 биткойна». Общая сумма вознаграждения, которую Цзин получает за майнинг блока, представляет собой сумму вознаграждения coinbase (25 новых биткойнов) и отчислений (0.09094928) за все транзакции, включенные в этот блок, как показано в примере 10.4.

Пример 10.4 ❖ Coinbase-транзакция

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1

{
    "hex" :
"010000000100000000000000000000000000000000000000000000000000000000000000ffffffff0
f03443b0403858402062f503253482fffffffff0110c08d9500000000232102aa970c592640d19de03ff
6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac00000000",
    "txid" : "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "coinbase" : "03443b0403858402062f503253482f",
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 25.09094928,
            "n" : 0,
            "scriptPubKey" : {
                "asm" :
"02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21OP_CHECKSIG",
                "hex" :
"2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
                "reqSigs" : 1,
                "type" : "pubkey",
                "addresses" : [
                    "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                ]
            }
        }
    ]
}
```

В отличие от обычных транзакций, coinbase-транзакция не потребляет (не расходует) данные UTXO как входные данные. В ней содержится только один

фрагмент входных данных, называемый coinbase, который создает биткойны из ничего. Coinbase-транзакция содержит один фрагмент выходных данных, осуществляющий оплату по биткойн-адресу майнера. Выходные данные coinbase-транзакции посылают значение, равное 25.09094928 биткойна, на биткойн-адрес майнера, в рассматриваемом примере это адрес `1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N`.

Вознаграждение coinbase и отчисления за транзакции

При создании coinbase-транзакции узел Цзина сначала вычисляет общую сумму отчислений за транзакции, суммируя все входные и выходные данные 418 транзакций, добавленных в этот блок. Отчисления вычисляются по следующей формуле:

```
Total Fees = Sum(Inputs) - Sum(Outputs)
```

В блоке 277316 общая сумма отчислений за транзакции составляет 0.09094928 биткойна.

Далее узел Цзина вычисляет корректную сумму вознаграждения за новый блок. Сумма вознаграждения вычисляется на основе высоты блока, начиная с исходной суммы в 50 биткойнов за блок, и сокращается наполовину через каждые 210 000 блоков. Поскольку рассматриваемый блок имеет высоту 277 316, действительная сумма вознаграждения равна 25 биткойнам.

Процесс вычисления можно проследить в функции `GetBlockSubsidy` в программном коде клиента Bitcoin Core, как показано в примере 10.5.

Пример 10.5 ❖ Вычисление вознаграждения за блок – функция GetBlockSubsidy, клиент Bitcoin Core, модуль main.cpp

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Установление нулевого вознаграждения, если бинарный сдвиг вправо не определен.
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // Вознаграждение уменьшается наполовину приблизительно через каждые 210000 блоков
    каждые 4 года.
    nSubsidy >>= halvings;
    return nSubsidy;
}
```

Начальная выплата вычисляется в сатоши с умножением на коэффициент 50 константы `COIN` (100 000 000 сатоши). Это действие устанавливает начальное вознаграждение (`nSubsidy`) равным 5 миллиардам сатоши.

После этого функция вычисляет количество выполненных сокращений наполовину `halvings` путем деления высоты текущего блока на интервал сокращений (`SubsidyHalvingInterval`). Для блока 277316 с интервалом сокращения наполовину через каждые 210 000 блоков получаем 1 сокращение.

Максимальное количество допустимых сокращений вдвое установлено равным 64, поэтому если количество сокращений превышает 64, то в коде определяется нулевое вознаграждение (возвращаются только отчисления за транзакции).

Далее функция использует оператор бинарного сдвига вправо для выполнения деления суммы вознаграждения (`nSubsidy`) на два для каждого периода сокращения наполовину. В рассматриваемом примере для блока 277316 операция бинарного сдвига вправо выполняется один раз для суммы вознаграждения в 5 миллиардов сатоши (одно сокращение), что дает в результате 2.5 миллиарда сатоши, или 25 биткойнов. Оператор бинарного сдвига вправо применяется, поскольку он наиболее эффективен для операции деления на два как целого числа, так и числа с плавающей точкой.

Наконец, вознаграждение за coinbase-транзакцию (`nSubsidy`) суммируется с отчислениями за транзакции (`nFees`), и полученная сумма возвращается как результат.

Если узел майнинга Цзина фиксирует coinbase-транзакцию, то что мешает Цзину «вознаградить» себя 100 или 1000 биткойнов? Ответ прост: некорректная сумма вознаграждения приведет к тому, что блок будет определен как некорректный всеми прочими узлами сети, и все затраты, в том числе и электроэнергия, израсходованная Цзином на доказательство выполнения работы (PoW), пропадут впустую. Цзин получит возможность расходования своего вознаграждения только в том случае, если блок будет принят всеми узлами сети.

Структура coinbase-транзакции

После всех описанных выше вычислений узел Цзина формирует coinbase-транзакцию для выплаты себе 25.09094928 биткойна.

Из примера 10.4 можно понять, что coinbase-транзакция имеет особый формат. Вместо входных данных транзакции, опредяляющих предыдущие данные UTXO для расходования, coinbase-транзакция содержит входные данные «coinbase». Входные данные транзакций были подробно описаны в табл. 6.2. Теперь сравним входные данные обычной транзакции с входными данными coinbase-транзакции. В табл. 10.1 показана структура обычной транзакции, а в табл. 10.2 описана структура coinbase-транзакции.

Таблица 10.1. Структура входных данных «обычной» транзакции

Размер	Поле	Описание
32 байта	Хэш транзакции	Указатель на транзакцию, содержащую расходуемые данные UTXO
4 байта	Индекс выходных данных	Номер индекса расходуемых данных UTXO, начиная с 0 (для первого фрагмента)
1–9 байтов (VarInt)	Размер разблокирующего скрипта	Длина разблокирующего скрипта (см. ниже) в байтах
Переменный	Разблокирующий скрипт	Скрипт, выполняющий условия скрипта, блокирующего данные UTXO
4 байта	Номер последовательности	В настоящее время неиспользуемая функциональная возможность Tx-replacement, установлено значение `0xFFFFFFFF`

Таблица 10.2. Структура входных данных coinbase-транзакции

Размер	Поле	Описание
32 байта	Хэш транзакции	Все биты нулевые: ссылка на хэш-значение транзакции отсутствует
4 байта	Индекс выходных данных	Все биты установлены в 1: 0xFFFFFFFF
1–9 байтов (VarInt)	Размер данных Coinbase	Длина данных Coinbase: от 2 до 100 байтов
Переменный	Данные Coinbase	Произвольные данные, используемые для дополнительных тегов nonce и майнинга. В блоках версии v2 это поле обязательно должно начинаться с высоты блока
4 байта	Номер последовательности	Установлено значение 0xFFFFFFFF

В coinbase-транзакции в первых двух полях установлены значения, не представляющие собой ссылки на данные UTXO. Вместо «хэш-значения транзакции» первое поле заполнено 32 нулевыми байтами. Поле «индекс выходных данных» (output index) содержит 4 байта с одинаковым значением 0xFF (255 в десятичной системе). Поле «Разблокирующий скрипт» (Unlocking Script; `scriptSig`) заменено на данные coinbase, а поле данных используется майнерами, как мы увидим ниже.

Данные coinbase

Coinbase-транзакции не содержат поля разблокирующего скрипта (`scriptSig`). Это поле заменено на данные coinbase, длина которых находится в диапазоне от 2 до 100 байтов. За исключением нескольких первых байтов, все прочие данные coinbase могут использоваться майнерами по их усмотрению, то есть это произвольные данные.

Например, в первичном блоке Сатоши Накамото добавил в данные coinbase текст «The Times 03/Jan/2009 Chancellor on brink of second bailout for banks», используемый как доказательство даты его создания и для передачи сообщения. В настоящее время майнеры пользуются данными coinbase для включения дополнительных значений nonce, а также строк, идентифицирующих конкретный пул майнинга.

Несколько первых байтов данных coinbase раньше тоже обычно использовалось произвольно, но сейчас положение изменилось. В соответствии с документом BIP-34 блоки version-2 (блоки, в поле версии которых записано значение 2) обязательно должны содержать индекс высоты блока (block height index) как скриптовую операцию `push` в самом начале поля coinbase.

В блоке 277316 мы видели, что данные coinbase (см. пример 10.4), располагающиеся на месте разблокирующего скрипта или поля `scriptSig` во входных данных транзакции, содержат шестнадцатеричное значение `03443b0403858402062f503253482f`. Попробуем расшифровать это значение.

Первый байт `03` сообщает механизму выполнения скриптов о необходимости записи следующих трех байтов в стек скрипта (см. табл. Б.1, приложение Б). Следующие три байта `0x443b04` – это высота блока, закодированная в формате «от младшего байта к старшему» (little-endian), то есть в обратном порядке:

самый младший байт записан первым. После изменения порядка байтов получаем результат 0x043b44, который в десятичном формате равен 277316.

Несколько следующих шестнадцатеричных цифр (0385840206) используется для кодирования дополнительного значения nonce (см. раздел «Дополнительное решение nonce» ниже в этой главе) или случайного значения, применяемого для поиска правильного решения задачи доказательства выполнения работы (PoW).

Завершающая часть данных coinbase (2f503253482f) – это строка в кодировке ASCII /P2SH/, которая сообщает, что узел майнинга, выполнивший майнинг этого блока, поддерживает среду P2SH, определенную в стандарте BIP-16. Введение функциональности P2SH потребовало от майнеров в явной форме определять используемый стандарт BIP-16 или BIP-17. Явным указанием на реализацию BIP-16 стало включение строки /P2SH/ в данные coinbase. Для обозначения реализации BIP-17 в соответствующие данные coinbase должна быть включена строка p2sh/CHV. Стандарт BIP-16 был объявлен «победителем», поэтому многие майнеры продолжали и продолжают включать строку /P2SH/ в свои данные coinbase для объявления поддержки именно этой функции.

В примере 10.6 используется библиотека libbitcoin, описанная в разделе «Прочие клиенты, библиотеки и инструментальные пакеты» главы 3, для извлечения данных coinbase из первичного блока и вывода сообщения Сатоши. Отметим, что библиотека libbitcoin содержит статическую копию первичного блока, поэтому в коде примера есть возможность извлечь первичный блок непосредственно из библиотечного кода.

Пример 10.6 ❖ Извлечение данных coinbase из первичного блока

```
/*
  Вывод сообщения Сатоши, записанного в первичном блоке.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Создание первичного блока
    const bc::block_type block = bc::genesis_block();
    // Первичный блок содержит единственную coinbase-транзакцию
    assert(block.transactions.size() == 1);
    // Получение первой транзакции в блоке (coinbase)
    const bc::transaction_type& coinbase_tx = block.transactions[0];
    // Coinbase tx содержит единственный фрагмент входных данных
    assert(coinbase_tx.inputs.size() == 1);
    const bc::transaction_input_type& coinbase_input = coinbase_tx.inputs[0];
    // Преобразование входных данных скрипта в собственный простейший ("сырой") формат
    const bc::data_chunk raw_message = save_script(coinbase_input.script);
    // Преобразование в формат std::string
    std::string message;
    message.resize(raw_message.size());
    std::copy(raw_message.begin(), raw_message.end(), message.begin());
```

```
    // Вывод сообщения из первичного блока
    std::cout << message << std::endl;
    return 0;
}
```

Код примера скомпилирован с помощью компилятора GNU C++, после чего полученный выполняемый файл был запущен, как показано в примере 10.7.

Пример 10.7 ❖ Компиляция и запуск примера satoshi-words

```
$ # Компиляция исходного кода
$  g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Запуск выполняемого файла
$ ./satoshi-words
^D◆◆<GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Формирование заголовка блока

Для формирования заголовка блока узел майнинга должен заполнить шесть полей, перечисленных в табл. 10.3.

Таблица 10.3. Структура заголовка блока

Размер	Поле	Описание
4 байта	Версия	Номер версии для отслеживания обновлений ПО/протокола
32 байта	Хэш предыдущего блока	Ссылка на хэш-значение предыдущего (родительского) блока в цепочке
32 байта	Корень дерева Меркле	Хэш-значение корня дерева Меркле, состоящего из транзакций этого блока
4 байта	Метка времени	Приблизительное время создания этого блока (в секундах по Unix-времени)
4 байта	Цель	Цель алгоритма Proof-of-Work для этого блока
4 байта	Nonce	Случайное значение (счетчик), используемое в алгоритме Proof-of-Work

На момент завершения майнинга блока 277316 номер версии, описывающий структуру блока, был равен 2, и этот факт кодируется в формате «от младшего байта к старшему» (little-endian) в 4 байтах в виде `0x02000000`.

Далее узел майнинга должен добавить хэш-значение предыдущего блока (Previous Block Hash – prevhash). Это хэш-значение заголовка блока 277315, то есть предыдущего блока, полученного из сети, который узел Цзина принял и выбрал в качестве родительского (parent) блока для своего блока-кандидата 277316. Хэш-значение заголовка блока 277315:

```
000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

Выбрав конкретный родительский блок, указанный в поле Previous Block Hash в заголовке блока-кандидата, Цзин подтверждает достоверность майнинга родителя, наращивая цепочку, которая теперь завершается этим принятым блоком. По существу, этим действием Цзин «голосует» за достоверность майнинга, формируя корректную цепочку с самым высоким уровнем сложности.

Следующий шаг – объединение всех транзакций в дерево Меркле, чтобы добавить итоговый корень дерева Меркле в заголовок блока. Coinbase-транзакция записывается как самая первая транзакция в блоке. После нее добавляются еще 418 транзакций, в итоге блок содержит 419 транзакций. Из раздела «Дерево Меркле» главы 9 нам известно, что в дереве обязательно должно содержаться четное число листьев-узлов, поэтому последняя транзакция дублируется, создавая 420 узлов, каждый из которых содержит хэш-значение одной транзакции. Затем хэш-значения транзакций попарно объединяются, формируя очередной уровень дерева, до тех пор, пока все транзакции не будут объединены в единственный узел в «корне» дерева. Корень дерева Меркле объединяет все транзакции в одно 32-байтовое значение, которое вы могли видеть как «корень дерева Меркле» в примере 10.3, а здесь мы воспроизведем его еще раз:

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

Затем узел майнинга Цзина добавляет 4-байтовую метку времени в Unix-формате, то есть представляющую собой количество секунд, прошедших с полуночи по Гринвичу UTC/GMT 1 января 1970 года. Значение метки времени 1388185914 равнозначно дате 27 декабря 2013 года, пятница, 23:11:54 UTC/GMT.

После этого узел Цзина заполняет описание цели, которое определяет требуемое доказательство выполнения работы (PoW), чтобы сделать блок корректным. Цель хранится в блоке как метрика «target bits», представляющая собой кодировку цели в формате «мантисса – показатель степени». В этом формате показатель степени представлен 1 байтом, за которым следуют 3 байта мантиссы (или коэффициента). Например, в блоке 277316 значение метрики target bits равно `0x1903a30c`. Первая часть `0x19` – шестнадцатеричный показатель степени, следующая часть `0x03a30c` – коэффициент. Концепция использования цели описана в разделе «Изменение цели для корректировки сложности» ниже в текущей главе, а представление метрики «target bits» рассматривается в разделе «Представление целевого значения» также в текущей главе.

Последнее поле – специальное значение nonce, инициализируемое нулем.

После заполнения всех перечисленных полей заголовок блока сформирован и можно начинать процесс майнинга этого блока. Теперь целью становится поиск значения nonce, позволяющего получить хэш-значение заголовка блока, меньшее, чем значение цели (target). Узел майнинга должен будет проверить миллиарды и даже триллионы возможных значений nonce, прежде чем будет найдено правильное значение nonce, удовлетворяющее требованиям.

Майнинг блока

После завершения формирования блока-кандидата узлом Цзина наступает время майнинга этого блока с помощью специализированной аппаратуры фермы майнинга для нахождения решения по алгоритму доказательства выполнения работы (PoW), чтобы сделать блок корректным и легальным. На протяжении всей книги мы рассматриваем криптографические хэш-функции

с различных точек зрения их использования в биткойн-системе. Хэш-функция SHA256 – это функция, применяемая в процессе майнинга биткойнов.

Проще говоря, майнинг – это процесс хэширования заголовка блока с изменением одного параметра, многократно повторяемый до тех пор, пока вычисленное хэш-значение не совпадет с заданным целевым значением. Результат выполнения хэш-функции невозможно предсказать заранее, также невозможно создать шаблон, по которому формируется заданное хэш-значение. Эта особенность хэш-функций означает, что единственным способом получения хэш-значения, совпадающего с заданным целевым значением, являются повторяемые многократно попытки подобрать с помощью изменения случайным образом входных данных до получения требуемого хэш-значения, появляющегося непредсказуемо.

Алгоритм доказательства выполнения работы (PoW)

Алгоритм хэширования принимает входные данные произвольной длины и генерирует детерминированный результат фиксированной длины, цифровой отпечаток входных данных. Для каждых конкретных входных данных получаемое в результате хэш-значение всегда будет одним и тем же. Это хэш-значение всегда можно легко вычислить и проверить, используя тот же самый алгоритм хэширования. Главной характеристикой криптографического алгоритма хэширования является то, что при любых вычислениях практически невозможно найти два различных фрагмента входных данных, которые в результате хэширования дают одинаковый отпечаток (совпадение называют коллизией (collision)). Следовательно, практически невозможно выбрать входные данные таким образом, чтобы получить требуемый отпечаток. Единственным способом является случайный перебор вариантов входных данных в попытках найти подходящие.

При использовании алгоритма SHA256 выходные данные всегда имеют длину 256 битов вне зависимости от размера входных данных. В примере 10.8 применяется интерпретатор языка Python для вычисления хэш-значения по алгоритму SHA256 для фразы «I am Satoshi Nakamoto».

Пример 10.8 ❖ Пример применения алгоритма SHA256

```
$ python

Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

В примере 10.8 показан результат вычисления хэш-значения для строки `"I am Satoshi Nakamoto"`: `5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e`. Это 256-битовое число представляет собой хэш (hash) или дайджест (digest) исходной строки и зависит от каждой ее части. При добавлении любой буквы, знака препинания или какого-либо иного символа будет получено другое хэш-значение.

Если мы начнем изменять строку, то вполне ожидаемо увидим на выходе совершенно различные хэш-значения. Попробуем добавлять в конец строки произвольное число, используя для этого простой скрипт на языке Python в примере 10.9.

Пример 10.9 ❖ Скрипт с использованием SHA256 для генерации хэш-значений при итерациях значения nonce

```
# пример итеративного изменения значения nonce во входных данных для алгоритма хэширования

import hashlib

text = "I am Satoshi Nakamoto"

# итерации nonce от 0 до 19
for nonce in xrange(20):
    # добавление значения nonce в конец текста
    input = text + str(nonce)

    # вычисление хэш-значения по алгоритму SHA-256 для входных данных (text+nonce)
    hash = hashlib.sha256(input).hexdigest()

    # вывод входных данных и вычисленного хэш-значения
    print input, '=>', hash
```

Выполнение этого примера позволяет генерировать хэш-значения для множества строк, которые сделаны различными при помощи добавления некоторого числа в конец текста. Последовательно увеличивая на единицу это добавляемое число, можно получать различные хэш-значения, как показано в примере 10.10.

Пример 10.10 ❖ Вывод скрипта, использующего SHA256 для генерации хэш-значений при итеративном изменении значения nonce

```
$ python hash_example.py

I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Для каждой новой строки вычисляется совершенно отличающееся от остальных хэш-значение. Хэш-значения кажутся абсолютно случайными, но полученные в этом примере результаты всегда можно точно воспроизвести на любом компьютере с установленным интерпретатором языка Python и увидеть те же самые хэш-значения.

Число, используемое в этом скрипте как переменное значение, называется nonce. Значение nonce применяется для изменения выходных данных криптографической функции, в нашем примере – для изменения SHA256-отпечатка для заданной строки.

Чтобы усложнить задачу с применением этого алгоритма, установим конкретную цель: найти строку, для которой генерируется шестнадцатеричное хэш-значение, начинающееся с нуля. К счастью, это не так уж сложно. В примере 10.10 можно видеть, что строка «I am Satoshi Nakamoto13» дает хэш-значение `0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5`, которое соответствует поставленному условию. Оно было найдено с 14-й попытки (после 13 неудачных). С точки зрения теории вероятности, если выходные данные хэш-функции распределяются равномерно, то нам следует ожидать появления результата с шестнадцатеричным префиксом 0 через каждые 16 хэш-значений (поскольку это одна из 16 шестнадцатеричных цифр от 0 до F). С точки зрения вычислительной математики это означает поиск хэш-значения, меньшего `0x1000000000000000000000000000000000000000000000000000000000000000`. Это значение называют пороговым значением (threshold) цели (target), а задача состоит в том, чтобы найти хэш-значение, которое в числовом выражении меньше цели. Если мы уменьшаем целевое (пороговое) значение, то задача поиска хэш-значения, которое меньше целевого, становится все более и более сложной.

Простой аналогией является игра, в которой участники многократно бросают пару кубиков (игральных костей), пытаясь получить число, меньшее определенного целевого значения. В первом раунде целевое значение равно 12. Если игрок не выбросил две шестерки, то он выиграл. В следующем раунде устанавливается целевое значение 11. Игроки должны выбросить 10 или меньше очков для победы, и это тоже несложная задача. Но через несколько раундов целевое значение снижается, скажем, до 5. Теперь больше половины бросков кубиков будет превышать целевое значение, следовательно, такие броски будут проигрышными. Чем ниже опускается целевое значение, тем больше бросков кубиков потребуется для выигрыша (экспоненциальный рост). В конце концов, когда целевое значение станет равным 2 (минимально возможным), только один бросок из 36, или 2%, позволит добиться победного результата.

С точки зрения стороннего наблюдателя, которому известно, что целевое значение игры в кости равно 2, если кто-то успешно выполнил победный бросок, то можно предположить, что для этого ему потребовалось в среднем 36 попыток. Другими словами, всегда можно оценить объем проделанной работы на основании сложности, определяемой целевым значением. Если алгоритм основан на детерминированной функции, такой как SHA256, то сами по себе

входные данные представляют доказательство (proof) того, что был выполнен определенный объем работы (work) для получения результата, не превышающего заданного целевого значения. Таким образом, мы получаем доказательство выполнения работы (Proof-of-Work).

Даже если каждая попытка дает случайный результат, вероятность любого возможного результата можно вычислить заранее. Следовательно, результат (выходные данные) при заданном уровне сложности предоставляет доказательство выполнения определенного объема работы.

В примере 10.10 «победившее» значение nonce равно 13, и этот результат может быть подтвержден любым «игроком», независимо от других. Любой желающий может добавить число 13 в конец строки «I am Satoshi Nakamoto», вычислить хэш-значение и проверить, действительно ли результат меньше целевого значения. Успешный результат также является доказательством выполнения работы, поскольку подтверждает, что действительно проделана работа по поиску этого значения nonce. Для проверки требуется лишь одно вычисление хэш-значения, в то время как для нахождения правильного значения nonce потребовалось 14 вычислений. Если установить более низкое целевое значение (сложность увеличивается), то потребуется намного больше вычислений хэш-значений, чтобы найти соответствующее значение nonce, но для проверки по-прежнему потребуется лишь одно вычисление в любом случае. Более того, зная целевое значение, любой может оценить уровень сложности, используя для этого статистические методы, следовательно, может узнать, какой объем работы потребовалось выполнить, чтобы найти правильное значение nonce.

Алгоритм доказательства выполнения работы (Proof-of-Work) должен генерировать хэш-значение, меньшее, чем целевое значение (target). Более высокое целевое значение определяет более низкий уровень сложности задачи поиска хэш-значения, меньшего целевого значения. Чем ниже целевое значение, тем выше уровень сложности поиска правильного хэш-значения. Между целевым значением и уровнем сложности установлена обратно пропорциональная зависимость.

Доказательство выполнения работы (Proof-of-Work) в биткойн-системе очень похоже на задачу, показанную в примере 10.10. Майнер формирует блок-кандидат, заполненный транзакциями. После этого майнер вычисляет хэш-значение заголовка этого блока и проверяет, является ли полученное значение меньшим, чем текущее установленное целевое значение (target). Если вычисленное хэш-значение не меньше целевого, то майнер изменяет значение nonce (обычно просто увеличивая его на единицу) и повторяет попытку. При текущем установленном уровне сложности в биткойн-сети майнеры должны выполнить квадриллионы попыток, прежде чем найдут значение nonce, при котором хэш-значение заголовка блока будет меньше целевого значения.

Весьма упрощенный вариант реализации алгоритма доказательства выполнения работы на языке Python показан в примере 10.11.

Пример 10.11 ❖ Упрощенная реализация алгоритма доказательства выполнения работы

```
#!/usr/bin/env python
# пример упрощенной реализации алгоритма proof-of-work

import hashlib
import time

max_nonce = 2 ** 32 # 4 billion

def proof_of_work(header, difficulty_bits):

    # вычисление сложности достижения цели
    target = 2 ** (256-difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()

      # проверка: получен ли правильный результат, меньший целевого значения
        if long(hash_result, 16) < target:
            print "Success with nonce %d" % nonce
            print "Hash is %s" % hash_result
            return (hash_result,nonce)

    print "Failed after %d (max_nonce) tries" % nonce
    return nonce


if __name__ == '__main__':

    nonce = 0
    hash_result = ''

# Уровень сложности от 0 до 31 бита
    for difficulty_bits in xrange(32):

        difficulty = 2 ** difficulty_bits
        print "Difficulty: %ld (%d bits)" % (difficulty, difficulty_bits)

        print "Starting search..."

        # контрольная точка: проверка текущего времени
        start_time = time.time()

        # содание нового блока, включающего хэш-значение из предыдущего блока
        # здесь выполняется имитация блока с транзакциями - заменяется обычной строкой
        new_block = 'test block with transactions' + hash_result

        # поиск требуемого правильного значения nonce для нового блока
        (hash_result, nonce) = proof_of_work(new_block, difficulty_bits)

        # контрольная точка: сколько времени потребовалось для нахождения результата
        end_time = time.time()

        elapsed_time = end_time - start_time
        print "Elapsed Time: %.4f seconds" % elapsed_time

        if elapsed_time > 0:

            # оценка скорости вычисления хэш-значений в секунду
            hash_power = float(long(nonce)/elapsed_time)
            print "Hashing Power: %ld hashes per second" % hash_power
```

При выполнении этого кода вы можете установить требуемый уровень сложности (в битах: сколько начальных битов должны быть нулевыми), после чего наблюдать, сколько времени потребуется на поиск решения. В примере 10.12 можно видеть, как предложенная реализация работает на ноутбуке со средними характеристиками (аппаратными ресурсами).

Пример 10.12 ❖ Выполнение примера реализации Proof-of-Work с различными уровнями сложности

```
$ python proof-of-work-example.py*

Difficulty: 1 (0 bits)

[...]

Difficulty: 8 (3 bits)
Starting search...
Success with nonce 9
Hash is 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Elapsed Time: 0.0004 seconds
Hashing Power: 25065 hashes per second
Difficulty: 16 (4 bits)
Starting search...
Success with nonce 25
Hash is 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Elapsed Time: 0.0005 seconds
Hashing Power: 52507 hashes per second
Difficulty: 32 (5 bits)
Starting search...
Success with nonce 36
Hash is 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hashes per second

[...]

Difficulty: 4194304 (22 bits)
Starting search...
Success with nonce 1759164
Hash is 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Elapsed Time: 13.3201 seconds
Hashing Power: 132068 hashes per second
Difficulty: 8388608 (23 bits)
Starting search...
Success with nonce 14214729
Hash is 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Elapsed Time: 110.1507 seconds
Hashing Power: 129048 hashes per second
Difficulty: 16777216 (24 bits)
Starting search...
Success with nonce 24586379
Hash is 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Elapsed Time: 195.2991 seconds
```

```
Hashing Power: 125890 hashes per second

[...]

Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291
Hash is 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per second
```

Можно видеть, что увеличение уровня сложности на 1 бит приводит к удвоению времени, затраченного на поиск решения. Если представить себе полное пространство 256-битовых чисел, то каждый раз при добавлении в условие одного бита, который должен быть нулевым, пространство поиска уменьшается вдвое. В примере 10.12 было выполнено 84 миллиона попыток вычисления хэш-значения для поиска значения nonce, при котором генерируется хэш-значение с 26 начальными нулевыми битами. Даже со скоростью вычислений, превышающей 120 000 хэш-значений в секунду, при этом на среднем ноутбуке требуется 10 минут для нахождения нужного решения.

На момент написания этой книги в сети выполнялись попытки найти блок, хэш-значение заголовка которого меньше следующего числа:

```
0000000000000000029AB9000000000000000000000000000000000000000000
```

В начале этого целевого значения довольно-таки много нулей, таким образом, приемлемый диапазон хэш-значений значительно уменьшен, следовательно, намного труднее подобрать корректное хэш-значение. В среднем это потребует более 1.8 септа-хэш-вычислений (тысяча миллиардов миллиардов хэш-вычислений) в секунду в сети, чтобы создать новый блок. Задача выглядит невыполнимой, но, к счастью, сеть обеспечивает 3 экза-хэш-вычисления в секунду (EH/сек; экза – 10^{18}), то есть предоставляет вычислительную мощность, достаточную для создания нового блока в среднем через каждые 10 минут приблизительно.

Представление целевого значения

В примере 10.3 мы видели, что в блоке содержится целевое значение, в терминологии биткойна называемое «целевыми битами» (target bits) или просто «битами» (bits). В блоке 277316 это значение равно `0x1903a30c`. Эта форма записи отображает целевое значение задачи доказательства выполнения работы (PoW) в формате коэффициент/показатель_степени, где первые две шестнадцатеричные цифры – показатель степени, а следующие шесть шестнадцатеричных цифр – коэффициент. Таким образом, в рассматриваемом блоке показатель степени равен `0x19`, а коэффициент – `0x03a30c`.

Для вычисления уровня сложности решения задачи по представлению целевого значения применяется следующая формула:

```
target = coefficient * 2^(8 * (exponent - 3))
```

Используя эту формулу и значения битов сложности (целевых битов), получаем:

```
target = 0x03a30c * 2^(0x08 * (0x19 - 0x03))^

=> target = 0x03a30c * 2^(0x08 * 0x16)^

=> target = 0x03a30c * 2^0xB0^
```

В десятичном формате это значение представлено числом:

```
=> target = 238,348 * 2^176^

=> target =
22,829,202,948,393,929,850,749,706,076,701,368,331,072,452,018,388,575,715,328
```

Возвращаясь к шестнадцатеричному формату, получаем:

```
=> target = 0x0000000000000003A30C0000000000000000000000000000000000000000000
```

Это означает, что корректный и легальный блок с высотой 277 316 представляет собой блок, хэш-значение заголовка которого меньше, чем целевое значение. В бинарном формате это число должно содержать более 60 начальных битов, установленных в нуль. При таком уровне сложности один майнер, вычисляющий 1 триллион хэш-значений в секунду (1 тера-хэш в секунду, или 1 TX/сек), смог бы находить решение только через каждые 8496 блоков, или в среднем через каждые 59 дней.

Изменение целевого значения для регулирования уровня сложности

Как нам уже известно, целевое значение определяет уровень сложности, следовательно, влияет на продолжительность поиска решения по алгоритму доказательства выполнения работы (PoW). Здесь неизбежно возникает вопрос: зачем регулируется уровень сложности, кто и каким образом его регулирует?

Блоки с биткойнами генерируются в среднем через каждые 10 минут. Это жизненный ритм, своего рода пульс биткойн-системы, которым обоснована частота выпуска денежных единиц и скорость окончательного подтверждения транзакций. Этот ритм должен оставаться постоянным не только в течение короткого промежутка времени, но и на протяжении нескольких десятилетий. Предполагается, что с течением времени достаточно быстро будет продолжаться рост мощности компьютеров. Более того, количество участников майнинга и используемых ими компьютеров также постоянно изменяется. Для сохранения времени генерации блока в среднем равным 10 минутам необходимо регулировать уровень сложности майнинга в соответствии с перечисленными выше изменениями. Фактически целевое значение доказательства выполнения работы является динамическим параметром, который периодически регулируется для сохранения 10-минутного интервала между созданием блоков. Проще говоря, целевое значение устанавливается таким образом, что-

бы текущая мощность майнинга позволяла получать результат – новый блок – с 10-минутным интервалом.

Но каким образом такое регулирование осуществляется в полностью децентрализованной сети? Изменение целевого значения выполняется автоматически и независимо на каждом узле сети. Через каждые 2016 блоков все узлы корректируют целевое значение для алгоритма доказательства выполнения работы (PoW). Формула для регулирования целевого значения включает время, затраченное на поиск последних 2016 блоков, которое сопоставляется с ожидаемым временем, равным 20 160 минут (2016 блоков умножаются на требуемый 10-минутный интервал). Вычисляется отношение между действительным и требуемым интервалами времени, и соответственно вычисленному результату выполняется корректировка (увеличение или уменьшение) целевого значения. Проще говоря, если в сети обнаруживается, что блоки создаются быстрее, чем через 10 минут, то уровень сложности увеличивается (целевое значение уменьшается). Если блоки создаются медленнее, чем ожидается, то уровень сложности снижается (целевое значение увеличивается).

Формулу можно записать в следующем виде:

```
Новая_цель = Старая_цель * (Реальное_время_для_последних_2016_блоков / 20 160 минут)
```

В примере 10.13 показан код, используемый в программном обеспечении клиента Bitcoin Core.

Пример 10.13 ❖ Изменение целевого значения для алгоритма Proof-of-Work – метод `CalculateNextWorkRequired()` в модуле pow.cpp

```
// Пошаговое регулирование предельного значения
 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;

 // Изменение целевого значения
 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;
```


Регулирование целевого значения выполняется через каждые 2016 блоков, но в действительности из-за ошибки определения диапазона в исходном коде клиента Bitcoin Core оно основано на суммарном времени предыдущих 2015 блоков (а не 2016, как должно быть). В результате регулирование приводит к небольшому завышению уровня сложности на 0.05%.

Параметры `Interval` (2016 блоков) и `TargetTimespan` (две недели, или 1 209 600 секунд) определены в модуле *chainparams.cpp*.

Чтобы избежать резкого изменения уровня сложности, при корректировке для целевого значения устанавливается ограничивающий коэффициент, равный четырем (4), который в одном цикле не должен быть превышен. Если требуется коэффициент больше четырех, то целевое значение корректируется с коэффициентом 4, но не больше. Любая дальнейшая корректировка будет выполняться в следующем регулировочном периоде, поскольку дисбаланс сохранится в течение интервала создания следующих 2016 блоков. Таким образом, значительные несоответствия между вычислительной мощностью операций хэширования и уровнем сложности потребуют нескольких 2016-блоковых циклов для достижения верного соотношения.

Уровень сложности майнинга блока биткойнов составляет приблизительно 10 минут работы всей сети в целом. Определение этого интервала основано на оценке времени, затраченного на майнинг предыдущих 2016 блоков, и корректируется через каждые 2016 блоков. Корректировка осуществляется посредством увеличения или уменьшения целевого значения.

Отметим, что целевое значение не зависит от количества транзакций или от значений (сумм) транзакций. Это означает, что общий объем вычислений хэш-значений, следовательно, и электроэнергия, затраченная на обеспечение защиты биткойнов, также полностью независимы от количества транзакций. Количество биткойнов может расти, увеличивая общий объем денежной массы, но при этом оставаться защищенным без какого-либо увеличения вычислительной мощности для хэширования по сравнению с текущим уровнем. Увеличение вычислительной мощности хэширования происходит под воздействием рыночных механизмов, то есть при появлении новых майнеров, вступающих в состязание за вознаграждение. Пока необходимая вычислительная мощность для хэширования находится под управлением майнеров, честно конкурирующих в борьбе за вознаграждение, этого вполне достаточно для предотвращения атак типа «захват» (takeover), следовательно, и для защиты биткойнов.

Уровень сложности майнинга тесно связан со стоимостью электроэнергии и с обменным курсом биткойна по отношению к валюте, используемой для оплаты электроэнергии. Высокопроизводительные системы майнинга, вероятно, почти так же эффективны, как нынешнее поколение кремниевых изделий, преобразующих электроэнергию в вычислительную мощность хэширования с максимально возможной интенсивностью. Главным фактором воздействия на рынке майнинга является стоимость одного киловатт-часа электроэнергии, затраченного на создание биткойна, поскольку этот фактор определяет рентабельность майнинга, следовательно, стимулы для выхода на рынок майнинга или ухода с него.

Успешный майнинг блока

Как мы видели ранее, узел Цзина сформировал блок-кандидат и подготовил его для майнинга. Цзин располагает несколькими аппаратными фермами (стойками) майнинга со специализированными для этой конкретной цели интегральным схемами, где сотни тысяч этих интегральных схем выполняют вычисления по алгоритму SHA256 в параллельном режиме с невероятными скоростями. Многие из этих специализированных устройств соединены с узлом Цзина по USB или через локальную сеть. Узел майнинга, работающий на десктоп-компьютере Цзина, передает заголовок сформированного блока специализированному оборудованию майнинга, которое начинает проверку триллионов значений nonce в секунду.

Приблизительно через 11 минут после начала майнинга блока 277316 одно из аппаратных устройств майнинга находит решение и отправляет его обратно на узел майнинга. При вставке в заголовок блока значение nonce 4215469401 дает хэш-значение блока:

```
00000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

которое меньше целевого значения:

```
0000000000000003A30C0000000000000000000000000000000000000000000000
```

Сразу же узел майнинга Цзина передает этот блок всем своим партнерам. Партнеры принимают, проверяют, затем распространяют новый созданный блок. При распространении нового блока по сети каждый узел добавляет его в собственную копию структуры данных блокчейна, наращивая ее высоту до 277 316 блоков. Как только узлы майнинга принимают и проверяют новый блок, они прекращают свою работу по поиску блока на этой высоте и немедленно начинают вычисление следующего блока в цепочке, используя блок Цзина как «родительский». Создавая поверх блока Цзина новый найденный блок, другие майнеры, по существу, «голосуют» всей своей вычислительной мощностью майнинга, подтверждая правильность и достоверность блока Цзина и цепочки, которую он наращивает.

В следующем разделе мы рассмотрим процесс, который каждый узел применяет для проверки корректности и достоверности блока и выбора самой длинной цепочки, создавая тем самым консенсус, позволяющий формировать децентрализованную структуру данных блокчейна.

Проверка корректности нового блока

Третий этап (шаг) в механизме консенсуса биткойн-системы – независимая проверка корректности и достоверности (валидация) каждого нового блока каждым узлом сети. По мере того как новый блок с вычисленным решением продвигается по сети, каждый узел выполняет ряд проверок корректности и достоверности (валидацию) блока, прежде чем передать его дальше следу-

ющим партнерам. Такой подход гарантирует, что в сети распространяются только корректные блоки. Независимая процедура проверки также гарантирует, что майнеры, действующие честно, заслуживают включения своих блоков в структуру данных блокчейна, то есть получают вознаграждение по заслугам. Блоки майнеров, действующих нечестно, отвергаются, и такие майнеры не только лишаются вознаграждения, но к тому же затрачивают впустую огромные усилия на поиск решения по алгоритму доказательства выполнения работы и терпят убытки от расхода электроэнергии без какой-либо компенсации.

Когда узел получает новый блок, он начинает процедуру его валидации, проверяя длинный список критериев, которым непременно должен соответствовать новый блок. При любых несоответствиях блок отвергается. Эти критерии можно найти в программном коде клиента Bitcoin Core в функциях `CheckBlock` и `CheckBlockHeader`, а их краткие описания приведены ниже:

- структура данных блока должна быть синтаксически корректной;
- хэш-значение заголовка блока должно быть меньше, чем целевое значение (установленное алгоритмом доказательства выполнения работы PoW);
- метка времени блока не должна превышать будущее время более чем на два часа (это допущение сделано с учетом возможных ошибок при вычислении текущего локального времени);
- размер блока не должен превышать установленного предела;
- первая транзакция (и только первая транзакция) должна представлять собой coinbase-транзакцию;
- все транзакции в блоке должны быть проверены на корректность (валидность) в соответствии со списком критериев, приведенным в разделе «Независимая верификация транзакций» ранее в текущей главе.

Независимая проверка (валидация) каждого нового блока каждым узлом сети гарантирует, что майнеры лишены возможности действовать обманным путем. В предыдущих разделах мы видели, что майнеры записывают в блок транзакции, за что вознаграждаются новыми биткойнами, созданными в этом же блоке, и заявляют суммы отчислений за включенные транзакции. Но почему бы майнерам не вписать самим транзакцию на тысячу биткойнов вместо корректного вознаграждения? Потому что каждый узел проверяет блоки на соответствие приведенным выше правилам. Некорректная coinbase-транзакция приведет к тому, что весь блок будет признан некорректным, в результате такой блок будет отвергнут, следовательно, заявленная транзакция никогда не станет частью реестра. Майнеры вынуждены формировать абсолютно корректные блоки на основе общепринятых правил, которые соблюдают все узлы сети, и выполнять их майнинг для нахождения правильного решения по алгоритму доказательства выполнения работы PoW. При этом узлы затрачивают огромное количество элетроэнергии в процессе майнинга, и если майнеры пытаются мошенничать, вся израсходованная

электроэнергия и трудозатраты пропадают впустую. Именно поэтому независимая валидация является основным компонентом механизма децентрализованного консенсуса.

Формирование и выбор цепочек блоков

Заключительным этапом работы механизма децентрализованного консенсуса в биткойн-системе являются формирование (сборка) цепочки блоков и выбор цепочки с наиболее убедительным доказательством выполнения работы. После того как узел проверил корректность нового блока, выполняется попытка формирования цепочки посредством присоединения нового блока к существующей структуре данных блокчейна.

Узлы поддерживают три набора блоков: блоки, включенные в основную структуру данных блокчейна, блоки, образующие различные ответвления основной цепочки блокчейна (вторичные цепочки), и блоки, не имеющие известных родителей в известных цепочках (блоки-сироты). Некорректные блоки отбрасываются сразу же после того, как обнаруживается несоблюдение одного из критериев валидации, следовательно, не включаются в какую-либо цепочку.

Основная цепочка (main chain) в произвольный момент времени представляет собой любую корректную (valid) цепочку блоков, с которой связано наиболее убедительное доказательство выполнения работы. В большинстве случаев это также цепочка с наибольшим количеством блоков, за исключением варианта, когда две цепочки имеют одинаковую длину, но одна из них обладает более солидным доказательством выполнения работы. Основная цепочка также содержит ветви с блоками, являющиеся «родными братьями» блоков в основной цепочке. Такие блоки хотя и валидны, тем не менее не являются частью основной цепочки. Они сохраняются для ссылок в будущем в том случае, если одна из ветвей станет расширением основной цепочки в рабочем порядке. В следующем разделе «Разветвления структуры данных блокчейна» мы более подробно рассмотрим, каким образом возникают вторичные цепочки как результат практически одновременного завершения процесса майнинга блоков на одинаковой высоте.

После приема нового блока узел пытается вставить такой блок в существующую структуру данных блокчейна. Узел проверяет в новом блоке поле «хэш-значение предыдущего блока», которое представляет собой ссылку на родительский блок. Затем узел пытается найти указанный родительский блок в существующей структуре данных блокчейна. Чаще всего родитель находится на «верхушке» основной цепочки, таким образом, рассматриваемый новый блок наращивает основную цепочку. Например, новый блок 277316 содержит ссылку на хэш-значение своего родительского блока 277315. На большинстве узлов, принимающих новый блок 277316, уже имеется блок 277315 на вершине своей основной цепочки, что позволяет им установить связь нового блока с родителем и нарастить свою версию цепочки.

В следующем разделе «Разветвления структуры данных блокчейна» мы увидим, что иногда новые блоки увеличивают длину цепочки, которая не является основной. В этом случае узел присоединяет новый блок к вторичной цепочке, наращивая ее, затем сравнивает объем проделанной работы вторичной цепочки с основной цепочкой. Если во вторичной цепочке общий объем выполненной работы больше, чем в основной цепочке, то узел переходит на вторичную цепочку, то есть выбирает ее в качестве новой основной цепочки, а старая основная цепочка становится вторичной – цепочки меняются местами (reconverge). Если это узел майнинга, то он начинает формирование очередного блока, наращивающего новую, более длинную (более «трудоемкую») цепочку.

Если получен корректный блок, но для него не найден родитель в существующих цепочках, то такой блок считается «сиротой» (orphan). Блоки-сироты сохраняются в специальном пуле, где они остаются до тех пор, пока не будет получен соответствующий родительский блок. После получения блока-родителя и присоединения его к одной из существующих цепочек ссылающийся на него блок-сирота может быть извлечен из пула сирот и воссоединен со своим родителем, становясь частью той же цепочки. Блоки-сироты обычно возникают, когда майнинг двух блоков был завершен с чрезвычайно малым интервалом относительно друг друга, но узлы получают их в обратном порядке (потомок раньше родителя).

При выборе корректной (валидной) цепочки с наибольшим суммарным объемом выполненой работы все узлы в конечном итоге достигают общего для всей сети консенсуса. Временные несоответствия между цепочками в конце концов разрешаются по мере выполнения все большего объема работы с наращиванием одного из возможных вариантов цепочек. Узлы майнинга «голосуют» своей вычислительной мощью, выбирая цепочку, которую будет наращивать очередной блок, полученный в результате майнинга. После майнинга нового блока и наращивания выбранной цепочки этот новый блок фактически представляет собой «голос», поданный узлом-источником.

В следующем разделе мы рассмотрим, каким образом разрешаются несоответствия между конкурирующими цепочками (ответвлениями) при независимом выборе цепочки с наибольшим общим объемом выполненной работы.

Разветвления структуры данных блокчейна

Поскольку блокчейн является децентрализованной структурой данных, различные его копии не всегда согласованы между собой. Блоки могут прибывать на различные узлы в разное время, в результате чего узлы «видят» текущее состояние структуры блокчейна по-разному. Чтобы решить эту проблему, каждый узел всегда выбирает и пытается наращивать ту цепочку блоков, в которой представлено наиболее убедительное доказательство выполнения работы, другими словами, наиболее длинную цепочку или цепочку с наибольшим общим объемом фактически выполненной работы. Суммируя объем работ, за-

фиксированный в каждом блоке цепочки, узел может вычислить общий объем работы, затраченный на создание этой цепочки. Поскольку все узлы выбирают цепочку с наибольшим общим объемом выполненной работы, глобальная биткойн-сеть в конечном итоге переходит в полностью согласованное состояние. Разветвления возникают как временные несоответствия между версиями структуры данных блокчейна, и эти несоответствия в конце концов устраняются по мере добавления новых блоков в одну из ветвей.

Разветвления структуры данных блокчейна, рассматриваемые в текущем разделе, возникают естественным образом как результат задержек при передаче данных в глобальной сети. Немного позже в этой же главе мы рассмотрим преднамеренно создаваемые разветвления.

На нескольких следующих схемах мы проследим развитие процесса разветвления цепочек, рассматривая всю сеть в целом. На схемах показано упрощенное представление биткойн-сети. Для наглядности различные блоки обозначены разными геометрическими фигурами (звезда, треугольник, треугольник вершиной вниз, ромб), распределенными по всей сети. Каждый узел сети изображен в виде небольшого кружка.

Каждый узел имеет собственную точку зрения на глобальную структуру данных блокчейна. После получения блоков от своих соседей каждый узел начинает обновление своей копии структуры данных блокчейна, выбирая цепочку с наибольшим суммарным объемом выполненной работы. Для наглядности каждый узел (кружок) содержит некоторую геометрическую фигуру, представляющую конкретный блок, который на этом узле определен в качестве текущей вершины основной цепочки. Если вы видите в кружке, обозначающем узел, звезду, это значит, что блок-«звезда» является вершиной основной цепочки блокчейна по мнению рассматриваемого узла.

На первой схеме (рис. 10.2) в сети принята единая точка зрения на структуру данных блокчейна, где блок-звезда представляет вершину основной цепочки.

Разветвление возникает, когда два блока-кандидата начинают состязаться за формирование самой длинной цепочки блокчейна. Это вполне нормальное явление – два майнера нашли решение по алгоритму доказательства выполнения работы PoW почти одновременно (в весьма коротком интервале времени). Так как оба майнера получили решение для собственных блоков-кандидатов, они немедленно начинают распространять свои блоки-«победители» среди ближайших соседей, которые сразу же передают эти блоки далее по всей сети. Каждый узел, принявший корректный блок, включает его в свою копию структуры данных блокчейна, увеличивая цепочку на один блок. Если в дальнейшем этот узел обнаруживает другой блок-кандидат, связанный с тем же родителем, он включает нового кандидата во вторичную цепочку. В результате некоторые узлы «видят» первым один блок-кандидат, тогда как другие узлы обнаруживают сначала другой блок-кандидат – формируются две конкурирующие версии структуры данных блокчейна.

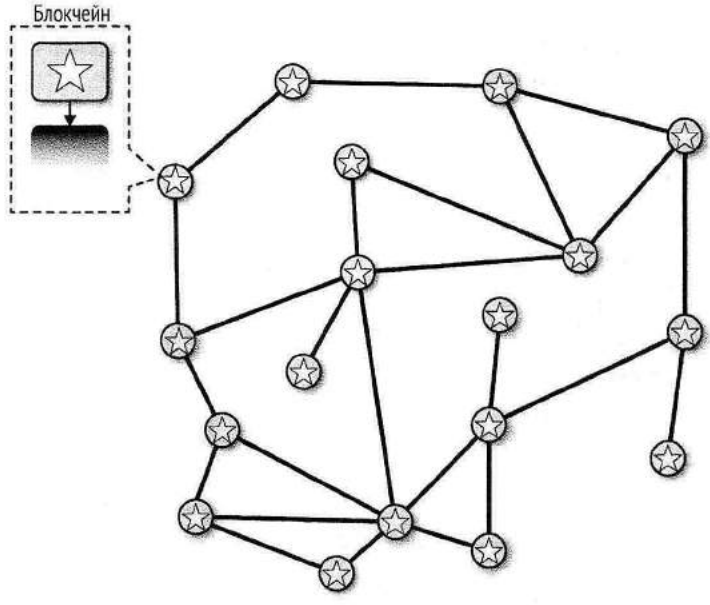


Рис. 10.2 ❖ Состояние перед разветвлением – все узлы имеют единую точку зрения

На рис. 10.3 мы видим двух майнеров (узел А и узел Б), сформировавших два различных блока-кандидата почти одновременно. Оба этих блока являются потомками блока-звезды и наращивают цепочку, размещаясь поверх блока-родителя (звезды). Для облегчения наблюдения за ними первый блок обозначен треугольником на узле-источнике А, второй блок обозначен треугольником с вершиной вниз на узле-источнике Б.

Теперь предположим, например, что узел майнинга А находит решение по алгоритму доказательства выполнения работы PoW для блока-«треугольника», наращивающего цепочку блокчейна поверх родительского блока-«звезды». Почти одновременно узел майнинга Б, также наращивающий цепочку блокчейна поверх родительского блока-«звезды», находит решение для блока-«треугольник вершиной вниз», своего блока-кандидата. Теперь существуют два блока-кандидата: один мы обозначили как «треугольник» с узла А, другой – «треугольник вершиной вниз» с узла Б. Оба блока корректны, оба содержат правильное решение по алгоритму доказательства выполнения работы PoW, оба наращивают цепочку относительно одного родителя (блок-«звезда»). Вероятнее всего, оба блока содержат в основном одни и те же транзакции, возможно, с небольшими отличиями в порядке их расположения.

По мере распространения этих двух блоков некоторые узлы принимают блок-«треугольник» первым, другие узлы сначала получают блок-«треугольник вершиной вниз». Как показано на рис. 10.14, в сети происходит

разделение по двум точкам зрения на структуру данных блокчейна: в одной части вершиной становится блок-«треугольник», в другой части – блок-«треугольник вершиной вниз».

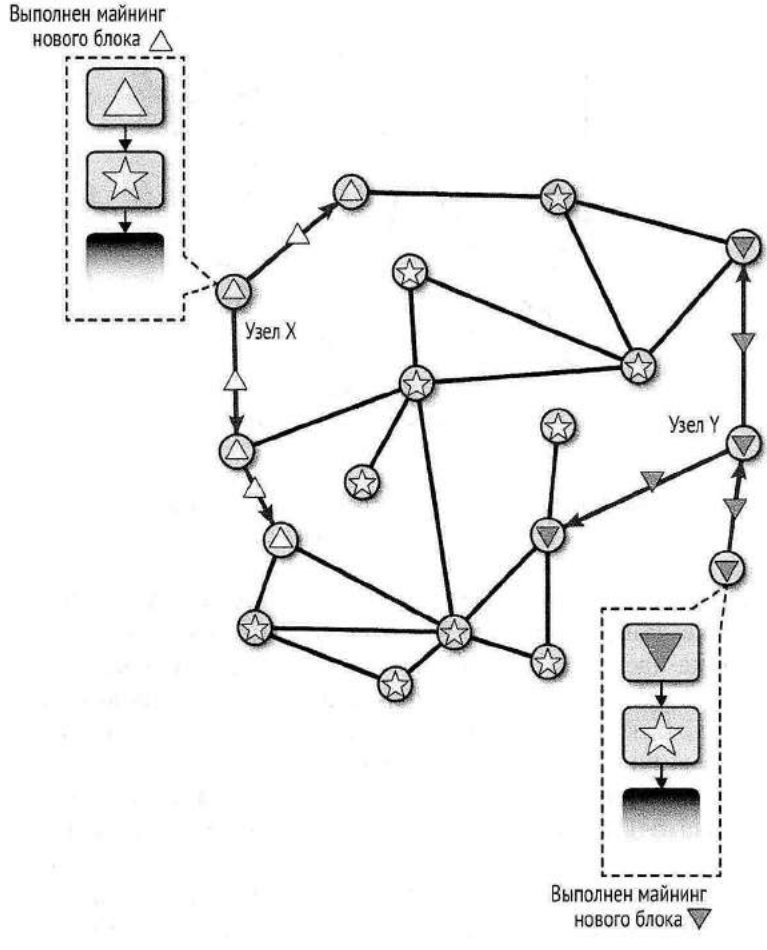


Рис. 10.3 ❖ Отображение события разветвления структуры данных блокчейна: два блока созданы почти одновременно

На последней схеме случайно выбранный «узел X» получает сначала блок-«треугольник» и с помощью этого блока наращивает свою цепочку поверх блока-«звезды». Узел X выбрал цепочку с блоком-«треугольником» как основную цепочку. Позже узел X получает блок-«треугольник вершиной вниз». Поскольку этот блок получен вторым, считается, что он «проиграл» состязание. Но блок-«треугольник вершиной вниз» не отбрасывается. Он также связыва-

ется с родительским блоком-«звездой» и формирует вторичную цепочку. Несмотря на то что узел X полагает, что правильно выбрал победившую цепочку, все же он сохраняет «проигравшую» цепочку, чтобы не потерять информацию, необходимую для того случая, когда «проигравшая» цепочка в итоге вдруг окажется «победившей».

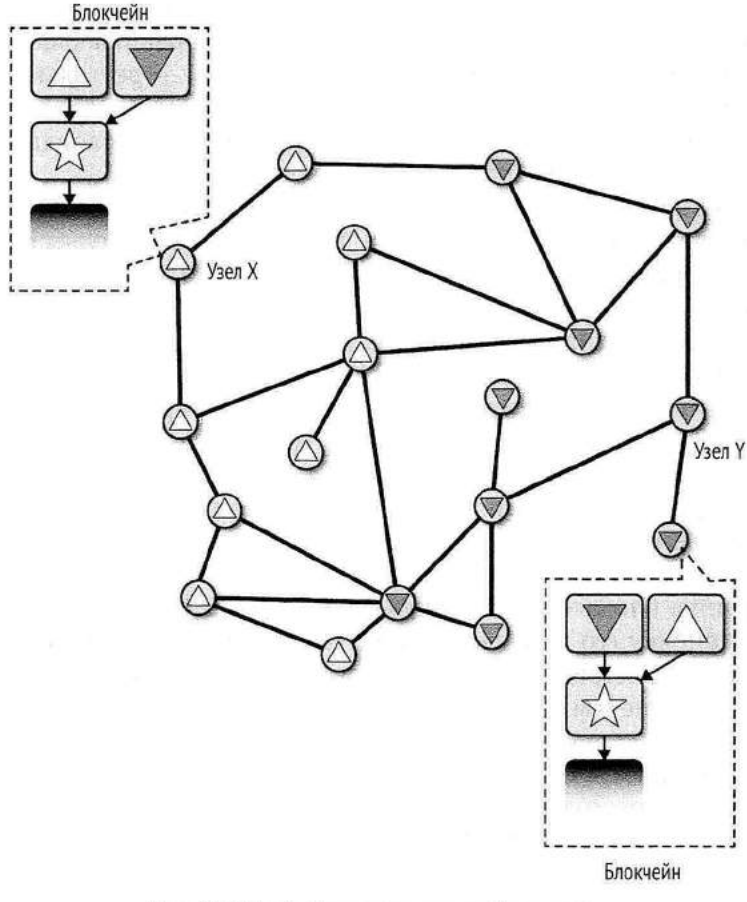


Рис. 10.14 ❖ Продолжение наблюдения за разветвлением структуры данных блокчейна: два блока распространяются по сети, разделяя ее на две части

В другой части сети узел Y формирует структуру данных блокчейна на основе собственной точки зрения на последовательность событий. Сначала он получает блок-«треугольник вершиной вниз» и выбирает цепочку с его участием как «победителя». В дальнейшем, когда узел Y получит блок-«треугольник», он присоединит его к родительскому блоку-«звезде», формируя вторичную цепочку.

Ни одну из двух частей сети нельзя считать ни «правильной», ни «неправильной». Обе части имеют вполне обоснованную и закономерную точку зрения на структуру данных блокчейна. Только после того, как одна из точек зрения станет преобладающей, можно в ретроспективе обосновать этот выбор, рассматривая процесс наращивания двух конкурирующих цепочек при выполнении дополнительной работы.

Узлы майнинга, точка зрения которых совпадает с точкой зрения узла X, немедленно начинают майнинг блока-кандидата, наращивающего цепочку с блоком-«треугольником» на вершине. Связывая блок-«треугольник» как родительский со своим блоком-кандидатом, они фактически голосуют за «треугольник» всей своей вычислительной мощью. Голоса этих узлов поддерживают цепочку, которую они выбрали в качестве основной.

Каждый узел майнинга, точка зрения которого совпадает с точкой зрения узла Y, начинает формирование блока-кандидата с родительским блоком-«треугольник вершиной вниз», наращивая цепочку, которую они считают основной. Таким образом, конкуренция возобновляется.

Проблема разветвления почти всегда решается в пределах одного блока. Часть сетевой вычислительной мощности направлена на создание нового блока поверх родительского блока-«треугольника», в то время как другая часть сетевой вычислительной мощности сосредоточена на продолжении цепочки поверх блока-«треугольника вершиной вниз». Даже если сетевая вычислительная мощность разделена почти поровну, вероятнее всего, одна группа майнеров найдет решение и начнет распространять его раньше, чем конкурирующая группа сможет найти какие-либо решения. Например, предположим, что майнеры группы блока-«треугольника» нашли новый блок-«ромб», наращивающий их цепочку (то есть звезда-треугольник-ромб). Майнеры немедленно начинают распространение нового блока, и вся сеть воспринимает это как правильное решение (см. рис. 10.5).

Все узлы, выбравшие блок-«треугольник» победителем предыдущего раунда, просто добавляют в основную цепочку еще один блок. Но узлы, которые предпочли блок-«треугольник вершиной вниз», теперь получают две цепочки: звезда-треугольник-ромб и звезда-треугольник_вершиной_вниз. Цепочка звезда-треугольник-ромб длиннее (больше общий объем выполненной работы), чем вторая цепочка. Поэтому такие узлы принимают цепочку звезда-треугольник-ромб в качестве основной, а звезда-треугольник_вершиной_вниз становится вторичной цепочкой, как показано на рис. 10.6. Происходит смена статуса цепочек (chain reconvergence), так как узлы второй группы вынуждены пересмотреть свою точку зрения на структуру данных блокчейна, чтобы принять новое явное доказательство в виде более длинной цепочки. Все майнеры, пытающиеся нарастить цепочку звезда-треугольник_вершиной_вниз, немедленно прекращают работу, потому что их блок-кандидат теперь стал «сиротой», а родительский блок-«треугольник вершиной вниз» уже не является вершиной самой длинной цепочки. Транзакции из блока-«треугольник вершиной вниз»

возвращаются в пул памяти с перспективой включения их в следующий блок, поскольку блок, содержащий эти транзакции, перестал быть частью основной цепочки. Вся сеть переходит к единой цепочке блокчейна звезда-треугольник-ромб, где «ромб» является самым последним блоком в основной цепочке. Все майнеры немедленно начинают работу с блоками-кандидатами, которые ссылаются на блок-«ромб» как на родительский, наращивая тем самым цепочку звезда-треугольник-ромб.

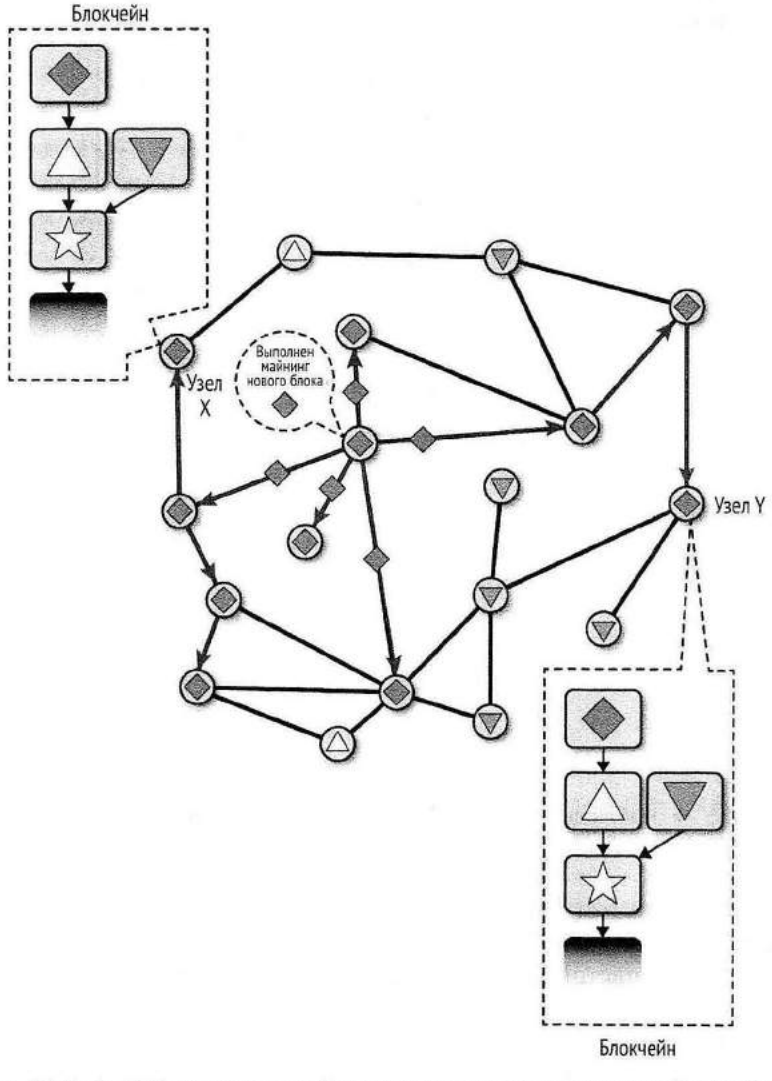


Рис. 10.5 ❖ Наблюдение за событиями разветвления цепочки блокчейна: новый блок наращивает одну из ветвей, заставляя сеть пересмотреть статус соответствующей цепочки

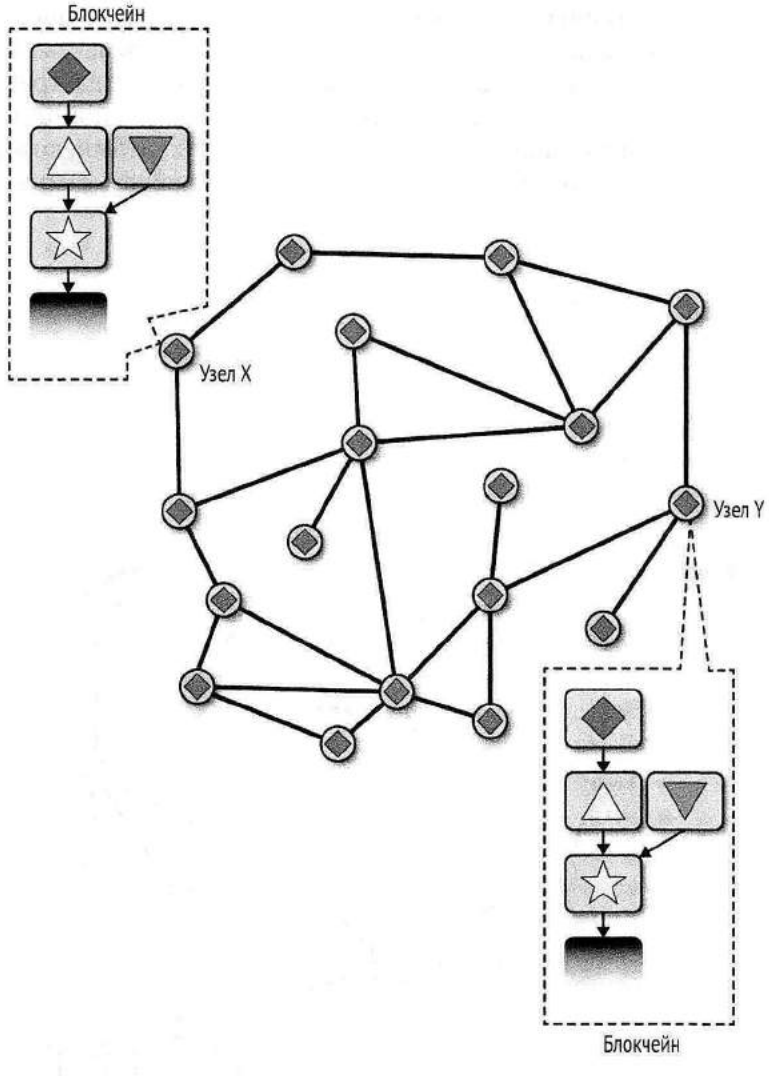


Рис. 10.6 ❖ Наблюдение за событиями разветвления цепочки блокчейна: вся сеть переходит на новую самую длинную цепочку

Теоретически возможно разветвление с наращиванием цепочки сразу по двум блокам, если эти два блока почти одновременно найдены майнерами «противоположных частей» относительно предыдущего разветвления. Но вероятность такого события крайне мала. Разветвление по одному блоку может происходить каждый день, тогда как разветвление по двум блокам встречается не чаще, чем один раз в две недели.

Интервал в 10 минут между созданием блоков биткойна представляет собой специально предусмотренный компромисс между краткими интервалами

времени подтверждения (расчеты по транзакциям) и вероятностью возникновения разветвления. Более короткий интервал создания блоков мог бы ускорить клиринг транзакций, но привел бы к более частым разветвлениям структуры данных блокчейна. Установленный в настоящее время более длинный интервал создания блоков сокращает количество разветвлений, но замедляет расчеты по транзакциям.

Майнинг и конкуренция в хэш-вычислениях

Майнинг биткойнов является областью исключительно высокой конкуренции. Вычислительная мощность операций хэширования возрастала по экспоненте в течение каждого года существования биткойнов. В некоторые годы рост отражал полную смену технологий, как, например, в 2010 и 2011 годах, когда многие майнеры перешли от использования обычных процессоров (CPU) к применению графических процессоров (GPU) и программируемых пользователем вентильных матриц (ППВМ – field-programmable gate array, FPGA) в процессе майнинга. В 2013 году появление интегральных схем специального назначения ASIC (application-specific integrated circuit) для майнинга привело к еще одному гигантскому скачку вычислительной мощности, поскольку позволило «зашить» реализацию алгоритма хэширования SHA256 непосредственно в чипы, предназначенные специально для майнинга. Первые такие чипы могли предоставить в одном корпусе бóльшую вычислительную мощность, чем вся биткойн-сеть в 2010 году.

В следующем списке показан рост общей вычислительной мощности биткойн-сети по отношению к первым пяти годам ее функционирования:

- 2009 год – 0.5 Мхэш/сек – 8 Мхэш/сек (16-кратный рост);
- 2010 год – 8 Мхэш/сек – 116 Гхэш/сек (14 500-кратный рост);
- 2011 год – 116 Гхэш/сек – 9 Тхэш/сек (562-кратный рост);
- 2012 год – 9 Тхэш/сек – 23 Тхэш/сек (2.5-кратный рост);
- 2013 год – 23 Тхэш/сек – 10 Пхэш/сек (450-кратный рост);
- 2014 год – 10 Пхэш/сек – 300 Пхэш/сек (3000-кратный рост);
- 2015 год – 300 Пхэш/сек – 800 Пхэш/сек (266-кратный рост);
- 2016 год – 800 Пхэш/сек – 2.5 Эхэш/сек (312-кратный рост).

График на рис. 10.7 показывает, что общая вычислительная мощность биткойн-сети увеличилась за последние два года. Здесь можно видеть, что конкуренция между майнерами и рост биткойн-системы привели к экспоненциальному увеличению вычислительной мощности для операций хэширования (общее количество операций хэширования за секунду во всей сети).

Поскольку общий объем вычислительных мощностей, применяемый для майнинга биткойнов, увеличился и продолжает увеличиваться с огромной скоростью, необходимо также увеличивать уровень сложности соответствующим образом. Метрика уровня сложности по графику на рис. 10.8 определяется как отношение текущего уровня сложности к минимальному уровню сложности (то есть уровню сложности самого первого блока).

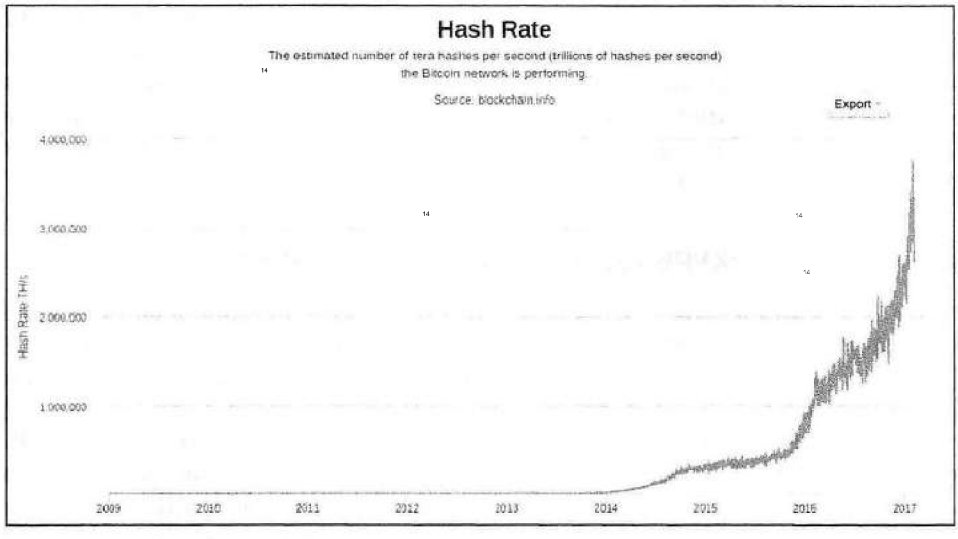


Рис. 10.7 ❖ Общая вычислительная мощность для операций хэширования, тера-хэш-операций в секунду (Тхэш/сек)

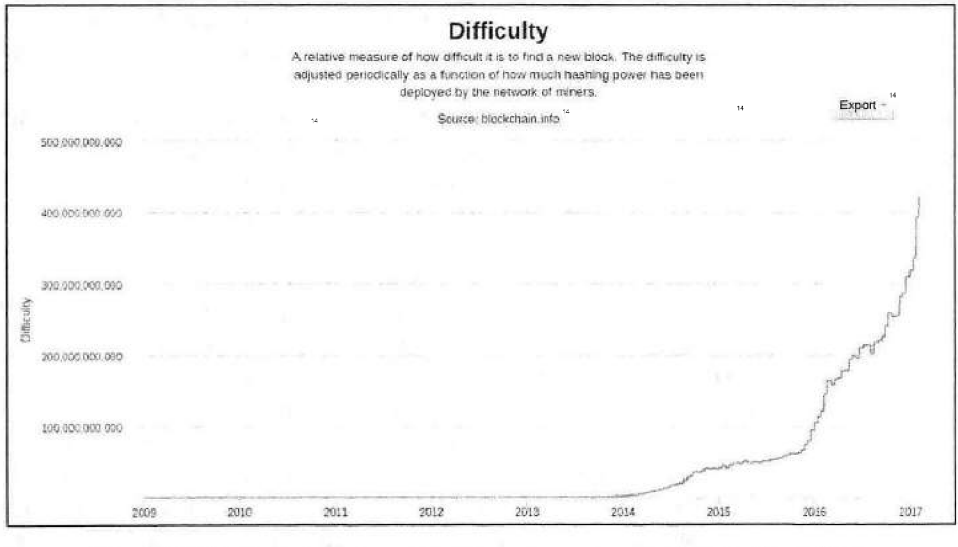


Рис. 10.8 ❖ Метрика уровня сложности процесса майнинга биткойнов

За последние два года внутренняя структура специализированных для майнинга ASIC-чипов существенно уплотнилась, достигая ныне существующего предельного размера (разрешения) при производстве кремниевых микросхем, равного 16 нанометрам (нм). В настоящее время производители ASIC-чипов намерены обогнать производителей обычных CPU-микросхем, проектируя

14-нанометровые чипы, поскольку рентабельность майнинга, определяющая развитие этой отрасли компьютерной индустрии, гораздо эффективнее, чем для компьютеров общего назначения. Теперь уже не наблюдается гигантских скачков в производительности майнинга биткойнов, так как эта отрасль достигла максимума, определяемого законом Мура (Moore's Law), утверждающего, что плотность элементов в чипе должна удваиваться приблизительно через каждые 18 месяцев[1]. Вычислительная мощность для майнинга в биткойн-сети продолжает расти по экспоненте, так как гонка за достижение более высокой плотности размещения элементов в микросхемах идет бок о бок с гонкой за более высокую насыщенность аппаратными средствами центров данных, где могут быть размещены тысячи таких микросхем. Теперь уже никого не интересует, какой объем майнинга можно выполнить на одном чипе, более важно, сколько чипов можно втиснуть в одно здание (или помещение) при постоянно сохраняющейся проблеме отведения тепла и обеспечения соответствующим уровнем электроэнергии.

Решение с расширением диапазона дополнительных значений nonce

С 2012 года майнинг биткойнов начал развиваться в направлении устранения основного ограничения в структуре заголовка блока. В начальном периоде развития биткойн-системы майнер мог найти блок с помощью итеративного перебора значений nonce для получения хэш-значения, меньшего, чем установленное целевое значение. По мере роста уровня сложности майнерам часто приходилось перебирать все 4 миллиарда значений nonce без обнаружения желаемого блока. Но эта проблема была легко устранимой с помощью обновления метки времени блока с учетом фактически затраченного на вычисления времени. Так как метка времени является частью заголовка, ее изменение позволяло майнерам повторять итеративный проход по значениям nonce с другими результатами. Но после того как мощность аппаратного оборудования для майнинга превысила 4 Гхэш/сек, сложность этой методики стала возрастать, поскольку перебор всех возможных значений nonce стал занимать менее секунды. После внедрения в процесс майнинга ASIC-оборудования и соответствующего повышения вычислительной мощности до нескольких Тхэш/сек для программного обеспечения майнинга потребовалось увеличение области допустимых значений nonce, чтобы обеспечить возможность нахождения корректных блоков. Можно было бы увеличить поле метки времени на один бит, но перемещение ее в слишком далекое будущее привело бы к тому,

---

[1] Здесь автор допустил неточность – закон Мура в современной формулировке: количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца. Часто цитируемый интервал в 18 месяцев связан с прогнозом Давида Хауса из Intel, по мнению которого *производительность* процессоров должна удваиваться каждые 18 месяцев. – *Прим. перев.*

что блок становился некорректным. Был необходим новый источник изменений в заголовке блока. Решением стало использование coinbase-транзакции как источника дополнительных значений nonce. Так как в coinbase-скрипте может сохраняться от 2 до 100 байтов данных, майнеры начали использовать этот «резерв» в качестве области дополнительных значений nonce, что позволило вести поиск в гораздо более широком диапазоне значений заголовка для поиска корректных блоков. Coinbase-транзакция включена в дерево Меркле, таким образом, любое изменение в coinbase-скрипте также изменяет корень дерева Меркле. Восемь байтов дополнительного значения nonce плюс 4 байта «стандартного» значения nonce позволяют майнерам исследовать в общей сумме 2^{96} (8 с 28 нулями) возможных значений в секунду без каких-либо изменений метки времени. Если в будущем майнеры смогут превысить и этот предел возможностей, то можно будет изменять метку времени. Кроме того, в coinbase-скрипте остается место для будущего расширения области допустимых значений nonce.

Пулы майнинга

В среде с чрезвычайно высокой конкуренцией отдельные майнеры (их также называют соло-майнерами) имеют весьма мало шансов на успех. Вероятность нахождения корректного блока одиноким майнером для компенсации своих расходов на электроэнергию и оборудование настолько мала, что напоминает попытку выиграть, участвуя в лотерее. Даже самая быстрая «потребительская» ASIC-система майнинга не может конкурировать на равных с коммерческими системами, состоящими из десятков тысяч таких чипов, объединенных в гигантские хранилища данных, специально расположенных рядом с гидроэлектростанциями. В настоящее время майнеры объединяются для создания так называемых пулов майнинга (mining pools), концентрируя свои вычислительные мощности и распределяя вознаграждение между тысячами участников. Участвуя в пуле, майнеры получают меньшую долю общего вознаграждения, но обычно вознаграждения поступают ежедневно, гарантируя определенную степень надежности.

Рассмотрим конкретный пример. Предположим, что майнер приобрел специализированное оборудование с суммарной мощностью 14 000 гигахэш в секунду (Гхэш/сек), или 14 Тхэш/сек. В 2017 году такое оборудование стоило приблизительно 2500 долларов США. Это оборудование в рабочем режиме потребляет 1375 ватт (1.3 КВт) электроэнергии, то есть 32 кВт-часов за сутки со стоимостью от 1 до 2 долларов за сутки по самым низким тарифам на электроэнергию. С учетом текущего уровня сложности в биткойн-системе майнер сможет в одиночку генерировать блок приблизительно лишь один раз в 4 года. Если майнер все же нашел корректный блок за этот интервал времени, он получит вознаграждение в 12.5 биткойна, что при обменном курсе около 1000 долларов за биткойн составит 12 500 долларов. Эта сумма недостаточна даже для компенсации расходов на приобретение оборудования и на электроэнергию,

потребленную за указанный период, причем чистый убыток составит приблизительно 1000 долларов. Но следует также учесть, что шанс нахождения желаемого блока в этот 4-летний период зависит еще и от удачливости майнера. Он может найти два блока за 4 года и получить весьма крупную прибыль. Или не найти ни одного блока даже за 5 лет и потерпеть значительные убытки. Еще более худшим для одинокого майнера является тот факт, что за указанный период уровень сложности решения по алгоритму доказательства выполнения работы PoW в биткойн-системе, вероятнее всего, существенно увеличится в соответствии с текущим ростом вычислительной мощности, следовательно, в распоряжении майнера остается не более одного года до того момента, когда его специализированное оборудование окончательно устареет и возникнет неизбежная необходимость его замены на более мощную аппаратуру майнинга. Но если такой майнер становится участником пула майнинга, то вместо томительного ожидания «упавших с неба» 12 500 долларов раз в четыре года при удачном стечении обстоятельств он получит гарантированную возможность зарабатывать приблизительно 50–60 долларов каждую неделю. Регулярные выплаты от пула майнинга помогут майнеру компенсировать затраты на оборудование и электроэнергию постепенно без излишнего риска. Разумеется, оборудование все равно будет устаревать за один или два года, и общая степень риска остается высокой, но, по крайней мере, обеспечиваются постоянный доход и надежность на этот период. С финансовой точки зрения такой подход имеет смысл только при очень низком тарифе на электричество (менее 1 цента за кВт) и только в крупном пуле.

Пулы майнинга координируют работу сотен или даже тысяч майнеров с помощью специализированных протоколов (pool-mining protocols). Отдельные майнеры конфигурируют свое оборудование для установления соединения с сервером пула после создания учетной записи в пуле. Во время майнинга оборудование поддерживает установленное соединение с сервером, синхронизируя свою работу с оборудованием других майнеров. Таким образом, майнеры в пуле разделяют между собой трудозатраты на майнинг блока и распределяют получаемые вознаграждения.

За успешно сгенерированные блоки вознаграждение выплачивается на биткойн-адрес пула, а не отдельным майнерам. Сервер пула периодически распределяет поощрительные платежи по биткойн-адресам майнеров, когда их доля в общей сумме вознаграждений достигает определенного порогового значения. Обычно сервер пула взимает некоторый процент отчислений с вознаграждений за предоставление сервиса по организации пула майнинга.

Между участниками пула распределяется работа по поиску решения для блока-кандидата, в ходе которой майнеры зарабатывают «доли» (shares) за вклад в общий процесс майнинга. Пул майнинга устанавливает более высокое целевое значение (то есть меньший уровень сложности) для зарабатывания долей, обычно более чем в 1000 раз проще, чем целевое значение биткойн-сети. Когда кто-то из участников пула завершает майнинг блока, вознаграждение

принимает пул, затем распределяет это вознаграждение между всеми майнерами пропорционально количеству долей, вложенных участниками в общий объем работы.

Пулы открыты для любых майнеров, больших и малых, профессионалов и любителей. Таким образом, в пуле некоторые участники могут иметь единственную небольшую машину майнинга, а у других отдельная комната или гараж заполнен самым современным специализированным оборудованием майнинга. Некоторые тратят на майнинг несколько десятков киловатт, другие запускают центры данных, потребляющие мегаватты электроэнергии. Каким образом пул майнинга измеряет и оценивает вклад отдельных участников, чтобы справедливо распределять вознаграждения без возникновения возможности мошенничества? Ответ все тот же: использование алгортима доказательства выполнения работы PoW биткойн-системы для количественной оценки вклада каждого майнера в пуле, но при этом устанавливается более низкий уровень сложности, чтобы даже самые «слабые» майнеры выигрывали долю вознаграждения достаточно часто, оправдывая тем самым свое участие в пуле. Устанавливая более низкий уровень сложности для зарабатывания долей, пул измеряет объем работы, выполненный каждым майнером. Каждый раз, когда майнер в пуле находит хэш-значение заголовка блока, меньшее, чем целевое значение пула, это служит реальным доказательством выполнения работы по хэшированию для вычисления требуемого результата. Но более важен тот факт, что работа по поиску долей вносит свой вклад в статистически измеримой форме в общие усилия по вычислению хэш-значения, меньшего, чем целевое значение биткойн-сети. Тысячи майнеров, пытающихся найти малые хэш-значения, в конечном итоге вычисляют результат, достаточно малый, чтобы соответствовать критерию целевого значения биткойн-сети.

Вернемся к аналогии игры в кости. Если игроки выбрасывают кости с целью получения результата, меньшего четырех (общий уровень сложности биткойн-сети), то пул должен установить более простое целевое значение, подсчитывая, сколько раз игрокам в пуле удалось выбросить результат, меньший восьми. Когда игроки получают результат, меньший восьми (целевое значение доли в пуле), они зарабатывают (выигрывают) доли, но не выигрывают игру в целом, так как не достигают общего целевого значения игры (сумма меньше четырех). Игроки будут достигать более простой цели пула гораздо чаще, весьма регулярно зарабатывая свои доли, даже если они не выполняют условия для более сложной цели всей игры. Время от времени один из игроков пула выбрасывает комбинацию костей с суммой, меньшей четырех, и пул выигрывает. Затем выигрыш может быть распределен между игроками пула на основе количества долей, которые они заработали. Даже несмотря на то, что цель «восемь или меньше» не являлась победной комбинацией, такой подход становится честным способом количественной оценки бросков игроков и рано или поздно дает выигрышную комбинацию «меньше четырех».

Аналогичным образом пул майнинга устанавливает собственное (более высокое и более простое) целевое значение, гарантирующее, что отдельный

участник пула достаточно часто сможет находить хэш-значение заголовка блока, которое меньше этого целевого значения пула, и заработать свои доли. Время от времени одна из попыток позволяет получить хэш-значение заголовка блока, которое меньше целевого значения биткойн-сети, что приводит к созданию корректного блока и выигрышу всего пула в целом.

Управляемые пулы

Большинство пулов является «управляемыми» (managed), то есть существует компания или отдельное лицо, организующее работу сервера пула. Владелец сервера пула называется оператором пула (pool operator) и выплачивает майнерам-участникам процентные отчисления от общего дохода.

На сервере пула работает специализированное программное обеспечение и поддерживается протокол пул-майнинга, координирующий действия участников. Кроме того, сервер пула соединен с одним или несколькими полноценными узлами биткойн-сети и имеет прямой доступ к полной копии базы данных блокчейна. Это позволяет серверу пула производить валидацию блоков и транзакций от лица майнеров пула, освобождая их от излишней нагрузки, связанной с поддержанием полноценного узла. Для участников пула это весьма важный факт, поскольку полноценный узел требует отдельного выделенного компьютера с дисковым пространством для постоянного хранения данных 100–150 Гб (как минимум) и по меньшей мере 2–4 Гб оперативной памяти (RAM). Более того, программное обеспечение полноценного узла требует постоянного наблюдения (мониторинга), сопровождения и частого обновления. Любой простой, вызванный ненадлежащим уровнем обслуживания или недостатком ресурсов, наносит значительный ущерб рентабельности оборудования узла майнинга. Для многих майнеров возможность участия в процессе майнинга без необходимости поддержки полноценного узла является еще одним большим преимуществом, стимулирующим желание присоединиться к управляемому пулу.

Майнеры-участники устанавливают соединение с сервером пула, используя протокол майнинга, например Stratum (STM) или GetBlockTemplate (GBT). Более старый стандарт GetWork (GWK) считается в большей степени устаревшим с конца 2012 года, так как не обеспечивает поддержку майнинга со скоростями, большими 4 Гхэш/с. Оба протокола STM и GBT создают шаблоны (templates) блока, содержащие шаблон (template) заголовка блока-кандидата. Сервер пула формирует блок-кандидат, объединяя в нем транзакции, добавляет coinbase-транзакцию (с пространством для дополнительных значений nonce), вычисляет корень дерева Меркле и устанавливает связь с хэш-значением предыдущего блока. Затем этот заголовок блока-кандидата передается каждому майнеру-участнику пула как шаблон. После этого каждый участник пула начинает майнинг, используя полученный шаблон блока, при более высоком (более простом) целевом значении, по сравнению с целевым значением биткойн-сети, и отправляет успешные результаты на сервер пула, чтобы заработать доли.

Пиринговый пул майнинга (P2Pool)

В управляемых пулах существует возможность мошенничества со строны оператора пула, который может направлять общие усилия пула на создание транзакций с двойным расходованием или на создание некорректных блоков (см. следующий раздел «Атаки на механизм консенсуса»). Более того, централизованные серверы пулов представляют собой единую точку отказа (single point of failure, SPOF). Если сервер пула приведен в неработоспособное состояние или замедлен в результате атаки типа DoS, то члены пула не смогут выполнять майнинг. В 2011 году для решения этих проблем, вызываемых централизацией, была предложена и реализована новая методика организации пула майнинга: P2Pool – пиринговый пул майнинга без центрального оператора.

Работа P2Pool основана на децентрализации функций сервера пула, то есть на реализации параллельной системы, организованной по образу и подобию структуры блокчейна, названной совместно используемой цепочкой (share chain). Совместно используемая цепочка – это структура блокчейна с уровнем сложности, меньшим, чем основная структура блокчейна биткойн-системы. Совместно используемая цепочка позволяет членам пула совместно работать в децентрализованном пуле, выполняя майнинг долей в общей цепочке со скоростью один долевой блок за каждые 30 секунд. В каждый блок совместно используемой цепочки записана пропорциональная доля вознаграждения для майнеров, внесших свой вклад в общую работу, с последовательным переносом долей из предыдущего долевого блока. Когда в одном из долевых блоков получен результат, соответствующий условию целевого значения биткойн-сети, этот блок распространяется и включается в структуру данных блокчейна биткойн-системы с вознаграждением всех майнеров, внесших вклад в поиск всех долевых блоков, предшествующих победившему долевому блоку. По существу, вместо сервера пула, отслеживающего доли и вознаграждения членов пула, совместно используемая (общая) цепочка позволяет всем майнерам в пуле прослеживать все доли, используя механизм децентрализованного консенсуса, подобного механизму консенсуса для структуры данных блокчейна биткойн-системы.

Майнинг в пуле P2Pool более сложен, чем майнинг в обычном пуле, поскольку он требует, чтобы майнеры работали на специально выделенном компьютере с достаточным дисковым пространством, объемом оперативной памяти и с установленным интернет-соединением с высокой пропускной способностью для поддержки полноценного биткойн-узла и специализированного программного обеспечения узла пула P2Pool. Майнеры пула P2Pool устанавливают соединение своего оборудования майнинга с локальным узлом P2Pool, который имитирует функции сервера пула, посылая шаблоны блоков на оборудование майнинга. В пуле P2Pool отдельные майнеры формируют собственные блоки-кандидаты, объединяя транзакции почти так же, как и независимые майнеры, но сам процесс майнинга выполняется совместно всеми членами пула в общей цепочке. Пул P2Pool представляет собой комбинированную методику, сочета-

ющую преимущества более мелких, но постоянных выплат вознаграждения, по сравнению с индивидуальными майнерами, но без необходимости передачи всей полноты управления оператору пула, в отличие от управляемых пулов.

Несмотря на то что P2Pool снижает уровень концентрации вычислительной мощности в руках операторов пулов майнинга, он остается потенциально уязвимым для атак типа «51%», направленных против самой совместно используемой цепочки. Более широкое распространение пулов P2Pool не решает проблему защиты от атаки типа «51%» всей биткойн-системы. Тем не менее пул P2Pool делает биткойн-систему в целом более надежной, поскольку представляет собой часть многообразной экосистемы майнинга.

Атаки на механизм консенсуса

Механизм консенсуса биткойн-системы, по крайней мере теоретически, уязвим для атак майнеров (или пулов), пытающихся использовать свою вычислительную мощность в мошеннических или разрушительных целях. Как мы видели ранее, механизм консенсуса зависит от наличия в биткойн-системе большинства майнеров, честно действующих в своих же интересах. Но если майнер или группа майнеров сможет получить в свое распоряжение значительную долю общей вычислительной мощности майнинга, то у них появляется возможность атаковать механизм консенсуса, разрушая защиту и доступность всей биткойн-сети.

Важно отметить, что атаки на механизм консенсуса могут повлиять только на будущий консенсус или, в лучшем случае, на самый последний прошлый консенсус (в пределах десятка блоков). С течением времени реестр биткойна становится все более и более неуязвимым для внесения изменений вручную. Хотя теоретически разветвление можно создать на любой глубине, на практике вычислительная мощность, необходимая для принудительного разветвления на очень большой глубине, настолько огромна, что делает старые блоки практически неизменяемыми. Атаки на механизм консенсуса никак не влияют на уровень защиты секретных ключей и алгоритма подписи (ECSDA). Атака на механизм консенсуса не может похищать биткойны, расходовать биткойны без подписи, перенаправлять биткойны или как-либо по-другому изменять ранее совершенные транзакции или записи о правах владения. Атаки на механизм консенсуса могут воздействовать только на самые последние блоки и вызывать затруднения, связанные с отказом от обслуживания (denial-of-service), при создании новых блоков.

Один из сценариев атаки на механизм консенсуса называется «атака 51%». По такому сценарию группа майнеров, управляющая большей частью (51%) общей вычислительной мощности биткойн-сети, вступает в тайный сговор для атаки на биткойн-систему. Обладая возможностью выполнения майнинга большинства блоков, майнеры-злоумышленники могут создавать преднамеренные разветвления в структуре данных блокчейна и транзакции с двойным

расходованием или осуществлять атаки типа DoS против конкретных транзакций и/или адресов. Атака с разветвлением/двойным расходованием – это ситуация, в которой атакующий искусственно делает ранее подтвержденные блоки некорректными, создавая разветвление ниже этих блоков и выполняя принудительный переход на альтернативную цепочку блоков. При наличии достаточной вычислительной мощности атакующий может сделать некорректными шесть и более смежных блоков, соответственно становятся некорректными и содержащиеся в них транзакции, которые до этого считались неизменяемыми (после шести подтверждений). Отметим, что двойное расходование возможно только в собственных транзакциях атакующего, для которых он способен предоставить корректную подпись. Собственные транзакции атакующего с двойным расходованием приносят пользу, если с помощью принудительного превращения транзакций в некорректные атакующий может получить невозвращаемую сдачу с некоторого платежа или товар без реальной его оплаты.

Рассмотрим практический пример атаки типа «51%». В первой главе мы изучали транзакцию между Алисой и Бобом с оплатой чашки кофе. Владелец кафе Боб хочет принимать платежи за порции кофе без ожидания их подтверждения (майнинга в блоке), поскольку риск двойного расходования стоимости чашки кофе весьма невелик, по сравнению с обеспечением удобства быстрого обслуживания клиентов. Это похоже на работу магазинов по продаже кофе, принимающих оплату кредитными картами без подписи для сумм, не превышающих 25 долларов, так как риск возврата денег на кредитную карту невелик, в то время как цена задержки транзакции при получении подписи более существенна. Но продажа товара с более высокой стоимостью за биткойны увеличивает риск атаки с двойным расходованием, когда покупатель распространяет еще и конкурирующую транзакцию, расходующую те же самые входные данные UTXO, и отменяет платеж продавцу. Атака с двойным расходованием может осуществляться двумя способами: или перед подтверждаемой транзакцией, или если атакующий имеет преимущественную возможность разветвления цепочки блокчейна посредством отмены нескольких блоков. Атака типа «51%» позволяет злоумышленникам дважды расходовать данные собственных транзакций в новой цепочке, соответственно отменяя транзакцию с теми же данными UTXO в старой цепочке.

В рассматриваемом примере атакующий злоумышленник Мэлори (Mallory) приходит в галерею Кэрол и покупает великолепную картину-триптих, изображающую Сатоши Накамото в образе Прометея. Кэрол продает Мэлори триптих «Пламенная страсть» (The Great Fire) за 250 000 долларов в биткойнах. Вместо того чтобы терпеливо дождаться шести и более подтверждений этой транзакции, Кэрол упаковывает и передает полотна Мэлори после одного подтвреждения. Мэлори работает вместе с сообщником Полом (Paul), который управляет крупным пулом майнинга. Сообщник Пол начинает атаку типа «51%» сразу же после включения траназкции Мэлори в блок. Пол направляет работу пула на повторный майнинг блока с той же высотой, что и блок, содержащий транзак-

цию Мэлори, заменяя платеж Мэлори владелице галереи Кэрол на транзакцию, выполняющую двойное расходование тех же самых входных данных, которые участвовали в первом платеже Мэлори. Транзакция с двойным расходованием потребляет эти же данные UTXO и возвращает их в кошелек Мэлори вместо отправки платежа Кэрол, то есть, по существу, позволяет Мэлори сохранить эти биткойны неизрасходованными. Затем Пол направляет работу пула на майнинг дополнительного блока, чтобы сделать цепочку, содержащую транзакцию с двойным расходованием, длиннее, чем исходная (корректная) цепочка (создавая цепочку ниже блока, содержащего транзакцию Мэлори). Когда разветвление цепочки блокчейна разрешается в пользу новой (более длинной) цепочки, транзакция с двойным расходованием заменяет первоначальный платеж, предназначенный для Кэрол. В результате Кэрол лишилась трех картин, не получив за них никакой оплаты в биткойнах. На протяжении всей этой операции участники пула майнинга Пола вполне могли оставаться в абсолютном неведении относительно попытки двойного расходования, поскольку процедура майнинга выполнялась автоматическими аппаратными средствами, из-за чего невозможно было проследить корректность каждой транзакции или блока.

Для защиты от атак этого типа коммерсант, продающий дорогостоящие товары, обязательно должен дождаться по меньшей мере шести подтверждений, прежде чем передать товар покупателю. Другой вариант – продавец должен использовать учетную запись с мультиподписями и с депонированием денежной суммы у третьего лица (в некотором роде – временное хранение денег), но и в этом случае он должен дождаться нескольких подтверждений, после чего депонированная сумма будет принята. Чем больше подтверждений произойдет, тем труднее сделать транзакцию некорректной с помощью атаки типа «51%». Для дорогостоящих товаров оплата биткойнами остается удобной и целесообразной, даже если покупателю приходится ждать доставки 24 часа, в течение которых происходят приблизительно 144 подтверждения.

Кроме атаки с двойным расходованием, существует вариант атаки на механизм консенсуса, вызывающий отказ в обслуживании (deny of service) для определенных участников биткойн-системы (для конкретных биткойн-адресов). Атакующий, обладающий большей частью вычислительной мощности, может просто игнорировать конкретные транзакции. Если такие транзакции включены в блок, формируемый другим майнером, атакующий может преднамеренно создать разветвление и выполнить майнинг альтернативного блока, опять же исключив те же самые транзакции. Этот тип атаки может привести к непрерывному отказу в обслуживании для конкретного биткойн-адреса или группы адресов на протяжении всего времени, пока атакующий сохраняет управление большей частью вычислительной мощности майнинга.

Несмотря на свое название, атака типа «51%» в действительности не требует 51% вычислительной мощности. На самом деле такую атаку можно попытаться предпринять с меньшим процентом от общей вычислительной мощности.

Пороговое значение 51% – это просто уровень, при котором атаке этого типа практически всегда гарантирован успех. Атака на механизм консенсуса, по существу, представляет собой «перетягивание каната» при создании очередного блока, и в этом состязании наиболее вероятна победа «более сильной» команды. С вычислительной мощностью, не превышающей 51%, вероятность успешной атаки снижается, так как другие майнеры управляют генерацией аналогичных блоков с помощью своей «честной» вычислительной мощности майнинга. Одну из точек зрения на эту проблему можно сформулировать так: чем большей вычислительной мощностью обладает атакующий, тем дольше он имеет возможность преднамеренно создавать разветвления цепочки, тем больше ранее созданных блоков он способен сделать некорректными или тем большим количеством блоков в будущем он сможет управлять. Группы исследования проблем безопасности использовали статистическое моделирование для выявления того факта, что различные типы атак на механизм консенсуса возможны при владении как минимум 30% общей вычислительной мощности.

Быстрый рост общей вычислительной мощности хэширования, вероятно, сделал биткойн-систему неуязвимой для атак одного майнера. Для майнера-одиночки не существует какой-либо возможности захвата управления сколько-нибудь значимой части общей вычислительной мощности майнинга. Но централизация управления, существующая в пулах майнинга, все же создает опасность проведения атак для получения прибыли оператором пула майнинга. Оператор управляемого пула полностью контролирует создание блоков-кандидатов и включение в них конкретных транзакций. Таким образом, для оператора пула возникает явная возможность исключения транзакций и/или добавления собственных транзакций с двойным расходованием. Если подобное злоупотребление властью совершается умело и в ограниченном объеме, то оператор пула может в течение долгого времени получать прибыль от атак на механизм консенсуса, оставаясь при этом незамеченным.

Но не все атакующие заинтересованы в получении прибыли. Одним из возможных сценариев атаки является намерение атакующего нарушить работу биткойн-сети без получения какой бы то ни было выгоды от этого разрушительного акта. Злонамеренная атака, нацеленная на нарушение работоспособности биткойн-системы, потребует огромных капиталовложений и секретного планирования, но, в принципе, вполне может быть осуществлена хорошо финансируемой, вероятнее всего, получившей всевозможную поддержку на государственном уровне атакующей группой. При другом варианте хорошо финансируемая атакующая группа может произвести атаку на механизм консенсуса биткойна с одновременным воздействием на концентрированные массивы оборудования майнинга, компрометацией операторов пулов и атаками типа «отказ в обслуживании» на прочие пулы. Все описанные выше сценарии теоретически возможны, но постепенно становятся нерациональными с практической точки зрения по мере постоянного экспоненциального роста общей вычислительной мощности хэширования биткойн-сети.

Вне всякого сомнения, серьезная атака на механизм консенсуса могла бы подорвать доверие к биткойн-системе на короткое время, возможно, вызвав значительное падение обменного курса биткойна. Но сама биткойн-сеть и соответствующее программное обеспечение постоянно развиваются и усовершенствуются, поэтому при атаках на механизм консенсуса биткойн-сообщество немедленно примет контрмеры, укрепляющие биткойн, делающие его более надежным и неуязвимым.

Изменение правил консенсуса

Правила консенсуса определяют корректность и законность (валидность) транзакций и блоков. Эти правила являются основой сотрудничества между всеми биткойн-узлами и отвечают за объединение всех локальных версий и вариантов в единую полностью согласованную структуру данных блокчейна для всей биткойн-сети в целом.

Несмотря на то что правила консенсуса считаются неизменяемыми на протяжении определенного короткого интервала времени и непременно должны быть полностью согласованными и едиными для всех узлов, их нельзя назвать неизменяемыми, если рассматривать достаточно длинный интервал времени. В целях развития и усовершенствования биткойн-системы правила непременно должны изменяться время от времени, чтобы соответствовать новым функциональным возможностям, новым технологическим достижениям или просто для исправления выявленных ошибок. Но, в отличие от общепринятой практики разработки программного обеспечения, обновления системы консенсуса связаны с намного более существенными затруднениями и требуют координации деятельности всех участников.

Устойчивые разветвления

Выше, в разделе «Разветвления структуры данных блокчейна», мы наблюдали за тем, как биткойн-сеть может ненадолго разделяться на две части, создающие две различные ветви цепочки блокчейна на короткое время. Мы видели, что этот процесс возникает естественным образом как составной элемент обычного функционирования сети, и убедились в том, что сеть всегда возвращается к единой общей структуре блокчейна после майнинга одного или нескольких блоков.

Существует и другой сценарий, при котором сеть может разделиться на две части, соответствующие двум цепочкам: изменение правил консенсуса. Этот тип разветвлений называется устойчивым разветвлением (hard fork), так как после такого разветвления сеть не выполняет преобразования в единую цепочку. Вместо этого две цепочки развиваются независимо друг от друга. Устойчивые разветвления возникают, когда часть сети работает под управлением набора правил консенсуса, отличающегося от набора правил консенсуса в остальной части сети. Такая ситуация может возникать из-за про-

граммной ошибки или вследствие преднамеренного изменения реализации правил консенсуса.

Устойчивые разветвления можно использовать для изменения правил консенсуса, но при этом потребуется координация действий всех участников системы. Все узлы, которые не выполнили обновления с целью перехода на новые правила консенсуса, лишаются возможности участвовать в работе механизма консенсуса и принудительно перемещаются в отдельную цепочку в момент устойчивого разветвления. Таким образом, изменения, вводимые устойчивым разветвлением, можно считать «несовместимыми снизу вверх» (not forward compatible), то есть с этого момента система без обновления не способна работать с новыми правилами консенсуса.

Рассмотрим механизм устойчивого разветвления на специальном примере.

На рис. 10.9 показана цепочка блокчейна с двумя разветвлениями. На высоте блока 4 возникает одноблоковое разветвление. Это тот тип самопроизвольного разветвления, который мы наблюдали ранее в разделе «Разветвления структуры данных блокчейна». После майнинга блока 5 сеть выполняет преобразование с переходом к единой цепочке, и проблема разветвления разрешается.

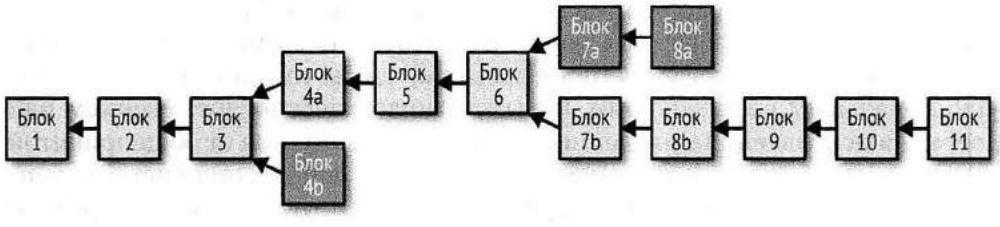


Рис. 10.9 ❖ Цепочка блокчейна с разветвлениями

Но позже, на высоте блока 6, возникает устойчивое разветвление (hard fork). Предположим, что вышла новая реализация клиента с изменениями правил консенсуса. Начиная с высоты блока 7 майнеры, начавшие работу с новой реализацией, принимают новый тип цифровой подписи – для примера назовем эту подпись «Smores», – основой которой уже не является алгоритм ECDSA. Сразу после этого узлы, запустившие новую реализацию, создают транзакцию, содержащую подпись Smores, а майнеры с обновленным программным обеспечением начинают майнинг блока 7b, содержащего эту транзакцию.

Любой узел или майнер, не обновивший программное обеспечение для проверки корректности новой цифровой подписи Smores, теперь не имеет возможности работать с блоком 7b. С их точки зрения, и транзакция с подписью Smores, и блок 7b, содержащий эту транзакцию, некорректны, так как в данном случае проверка основана на старых правилах консенсуса. Такие узлы отбрасывают блок 7b и транзакцию в нем, то есть не будут их распространять по сети. Майнеры, пользующиеся старыми правилами, не принимают блок 7b и продолжают майнинг блока-кандидата, родителем которого является блок 6.

В действительности майнеры, работающие по старым правилам, могут даже не получить блок 7b, если все узлы, с которыми они соединены, также продолжают использование старых правил, следовательно, не распространяют этот блок. В результате у таких узлов появляется возможность выполнить майнинг блока 7a, являющегося корректным по старым правилам и не содержащего транзакций с подписями Smores.

Разветвление двух цепочек продолжается и в дальнейшем. Майнеры в цепочке «b» продолжают принимать и майнить транзакции, содержащие подписи Smores, а майнеры в цепочке «a» продолжают игнорировать такие транзакции. Даже если блок 8b не содержит ни одной транзакции с подписью Smores, майнеры из цепочки «a» не смогут его обработать. Для них он выглядит как блок-сирота, так как его родительский блок 7b не распознан как корректный.

Устойчивые разветвления: ПО, сеть, майнинг и цепочка

Для разработчиков программного обеспечения термин «разветвление» (fork) имеет другой смысл, а термин «устойчивое разветвление» (hard fork) только добавляет путаницу[1]. В сфере разработки программного обеспечения с открытыми исходными кодами разветвление происходит, когда группа разработчиков выбирает другой план развития («дорожную карту») ПО и приступает к альтернативной реализации проекта с открытым исходным кодом. Мы уже рассматривали два условия, которые приводят к устойчивому разветвлению: программная ошибка в правилах консенсуса и преднамеренное изменение правил консенсуса. В случае преднамеренного изменения правил консенсуса разветвление версий программного обеспечения предшествует устойчивому разветвлению. Но для возникновения этого типа устойчивого разветвления обязательно должна быть разработана, принята как основная и запущена в эксплуатацию новая программная реализация правил консенсуса.

Примерами разветвления версий программного обеспечения с попытками изменения правил консенсуса являются Bitcoin XT, Bitcoin Classic и самая свежая версия Bitcoin Unlimited. Тем не менее ни одно из этих разветвлений версий ПО не привело к устойчивому разветвлению. Несмотря на то что разветвление версий ПО является необходимым предварительным условием, само по себе оно недостаточно для возникновения устойчивого разветвления. Для создания устойчивого разветвления альтернативная реализация ПО и определяемые ею новые правила должны быть приняты и активированы майнерами, кошельками и промежуточными узлами. Но, с другой стороны, существуют многочисленные альтернативные реализации Bitcoin Core и даже программные «форки», которые не изменяют существующих правил консенсуса, за исключением исправления явных ошибок, и могут совместно работать в сети и взаимодействовать друг с другом без создания устойчивых разветвлений.

---

[1] Следует отметить, что даже русскоязычные программисты гораздо чаще используют кальку «форк» для обозначения ответвления ПО.

Правила консенсуса могут различаться в явной и очевидной форме в процедуре валидации (проверки корректности и легальности) транзакций и блоков. Кроме того, правила консенсуса могут иметь не столь явные и не сразу заметные различия в конкретных реализациях правил, когда они применяются к биткойн-скриптам или к простейшим криптографическим объектам, таким как цифровые подписи. Наконец, правила консенсуса могут различаться совершенно непредвиденным образом из-за неявно подразумеваемых ограничений консенсуса, определяемых системными ограничениями или деталями реализации. Для последнего варианта примером может служить обнаруженное непредусмотренное устойчивое разветвление при обновлении Bitcoin Core с версии 0.7 на версию 0.8, которое было вызвано ограничениями реализации ПО Berkley DB, используемого для хранения блоков.

С теоретической точки зрения мы можем интерпретировать устойчивое разветвление как разработку в четырех стадиях: программное разветвление (разветвление версий ПО), разветвление сети, разветвление майнинга и разветвление цепочки.

Процесс начинается, когда разработчики создают альтернативную реализацию клиента с измененными правилами консенсуса.

После развертывания этой альтернативной реализации в сети определенный процент майнеров, кошельков пользователей и промежуточных узлов может принять и начать работу с этой реализацией. Полученное в результате разветвление будет зависеть от того, применимы ли новые правила консенсуса к блокам, транзакциям и некоторым другим аспектам системы. Если новые правила консенсуса относятся к транзакциям, то кошелек, создающий транзакцию по новым правилам, может неосмотрительно и слишком быстро создать разветвление сети, за которым последует и устойчивое разветвление, когда эта транзакция в процессе майнинга будет включена в блок. Если новые правила относятся к блокам, то процесс устойчивого разветвления начнется при майнинге блока по новым правилам.

Сначала произойдет разветвление сети. Узлы, использующие первоначальную реализацию правил консенсуса, будут отбрасывать все транзакции и блоки, созданные по новым правилам. Более того, узлы, соблюдающие старые правила консенсуса, будут временно блокировать и разрывать соединения со всеми узлами, которые посылают им эти «некорректные» транзакции и блоки. В результате сеть разделится на две части: старые узлы сохранят соединения только со старыми узлами, а новые узлы будут соединены только с новыми узлами. Единственная транзакция или блок на основе новых правил вызывает «волнение» во всей сети и в итоге делит ее на две части (на две сети).

После того как майнер, использующий новые правила, выполнит майнинг блока, вычислительная мощность майнинга и цепочка блоков также разделяются. Новые майнеры создают следующие блоки поверх этого нового блока, тогда как старые майнеры продолжают майнить отдельную цепочку на основе старых правил. Разделение сети создает ситуацию, при которой майнеры, работающие по различным правилам консенсуса, вероятнее всего, не будут полу-

чать блоков из «лагеря оппонентов», как если бы они были подключены к двум отдельным сетям.

Разделение майнеров и уровень сложности

После того как майнеры разделили процесс майнинга по двум различным цепочкам, вычислительная мощность хэширования также разделяется по этим цепочкам. Вычислительная мощность майнинга может быть разделена в любой пропорции между отдельными цепочками. Новые правила могут поддерживаться лишь небольшим объемом вычислительной мощности майнинга или получить поддержку большей части вычислительной мощности.

Предположим, например, что разделение произошло в пропорции 80% : 20%, то есть большая часть вычислительной мощности майнинга используется для поддержки новых правил консенсуса. Также предположим, что разветвление произошло сразу после завершения периода изменения целевого значения.

Каждая из двух цепочек должна наследовать уровень сложности из периода изменения целевого значения. Новые правила консенсуса имеют в своем распоряжении 80% ранее доступной общей мощности майнинга. С точки зрения «новой» цепочки, мощность майнинга неожиданно уменьшилась на 20%, по сравнению с прошедшим периодом. Блоки будут генерироваться в среднем через каждые 12 минут с учетом 20-процентного уменьшения вычислительной мощности майнинга, доступной для наращивания этой цепочки. Такая скорость создания блоков будет сохраняться (если не произойдет каких-либо изменений вычислительной мощности хэширования) до тех пор, пока не будет выполнен майнинг 2016 блоков, который займет приблизительно 24 192 минуты (при скорости один блок за 12 минут), или 16.8 суток. По истечении 16.8 суток произойдет изменение целевого значения и уровень сложности будет отрегулирован (уменьшен на 20%) для восстановления 10-минутного интервала создания новых блоков с учетом снижения объема вычислительной мощности в этой цепочке.

В другой цепочке с майнингом по старым правилам и всего лишь 20% вычислительной мощности майнинга задача будет намного более трудной. Теперь в этой цепочке блоки будут создаваться в среднем через 50 минут. Уровень сложности не будет регулироваться до тех пор, пока не сгенерируются 2016 блоков, а это произойдет через 100 800 минут, или приблизительно через 10 недель майнинга. Если предположить, что емкость блока постоянна, то при таких условиях скорость обработки транзакций снизится в 5 раз, так как станет гораздо меньше новых блоков (необходимо около часа на создание нового блока), доступных для записи транзакций.

Спорные устойчивые разветвления

Разработка программного обеспечения для механизма консенсуса пока еще находится на ранней стадии развития. Точно так же, как в начале процесса развития разработки программного обеспечения с открытыми исходными

кодами изменялись методы и технологии создания программного обеспечения, создавались новые методики, новые инструментальные средства и новые сообщества, так и в начальной стадии развития разработки программного обеспечения для механизма консенсуса формируется новая передовая отрасль информационных технологий. В результате обсуждений, экспериментов и преодоления препятствий мы увидим новые инструментальные средства разработки, практических методов, технологических методик и появление новых сообществ.

Устойчивые разветвления (hard forks) оцениваются как рискованные, так как они вынуждают меньшинство либо обновить ПО, либо оставаться в старой цепочке (поддерживаемой меньшинством). Опасность разделения всей системы на две конкурирующие системы в большинстве случаев оценивается как неприемлемый риск. В результате многие разработчики возражают против использования механизма устойчивых разветвлений для реализации обновлений правил консенсуса, если не будет обеспечена почти единодушная поддержка этого средства со стороны всей сети. Любые предложения по механизму устойчивых разветвлений, которые не получают «почти единодушной поддержки», считаются слишком «спорными» (contentious), чтобы уменьшить риски разделения системы.

Проблема устойчивых разветвлений вызывает бурные дискуссии в сообществе разработчиков биткойна, особенно если обсуждается тема возможных предлагаемых изменений в правилах консенсуса, управляющих ограничением на максимальный размер блока. Некоторые разработчики возражают против любой формы устойчивых разветвлений, считая их слишком опасными. Другие рассматривают механизм устойчивых разветвлений как важнейший инструмент обновления правил консенсуса способом, который позволяет избежать «унаследованных технических ограничений» и обеспечивает окончательный разрыв с прошлым. Еще одна группа разработчиков определяет устойчивые разветвления как механизм, который должен использоваться весьма редко с тщательным и подробным предварительным планированием и только при «почти единодушном консенсусе».

Ранее мы рассматривали появление новых методик для снижения рисков при устойчивых разветвлениях. В следующем разделе мы рассмотрим неустойчивые разветвления, а также методики BIP-34 и BIP-9 для оповещения и активации изменений механизма консенсуса.

Неустойчивые разветвления

Не все изменения в правилах консенсуса приводят к устойчивым разветвлениям. Только те изменения механизма консенсуса, которые не являются совместимыми снизу вверх, вызывают разветвление. Если изменение реализовано таким способом, что необновленный клиент продолжает оценивать транзакции и блоки как корректные по предыдущей версии правил, то изменение может произойти без разветвления.

Термин «неустойчивое разветвление» (soft fork) был введен, чтобы отличить описанный выше способ обновлений от случаев «устойчивого разветвления». С практической точки зрения неустойчивое разветвление вообще не является разветвлением. Неустойчивое разветвление – это совместимое снизу вверх («вперед» – forward-compatible) изменение правил консенсуса, которое позволяет необновленным клиентам продолжать работу с механизмом консенсуса после ввода новых правил.

Одна из особенностей неустойчивых разветвлений состоит в том, что обновления, используемые для ограничения правил консенсуса, а не для их расширения, заметны не сразу. Для соблюдения совместимости снизу вверх транзакции и блоки, создаваемые по новым правилам, обязательно должны быть корректными (валидными) и по старым правилам, но не наоборот. В новых правилах могут быть введены только ограничения по условиям валидности, в противном случае, если блок не будет принят по старым правилам, возникнет устойчивое разветвление.

Неустойчивые разветвления могут быть реализованы различными способами – сам термин определяет не какой-либо единственный метод, а целую группу методов, которые обладают одним общим свойством: они не требуют обязательного обновления всех узлов и не исключают принудительно не обновленных узлов из процедуры консенсуса.

Неустойчивые разветвления переопределяют коды NOP

В биткойн-системе реализовано несколько вариантов неустойчивых разветвлений на основе альтернативной интерпретации кодов операций NOP. В языке Bitcoin Script для использования в будущем зарезервированы десять кодов операций: от NOP1 до NOP10. По правилам консенсуса наличие этих кодов в скрипте интерпретируется как оператор, не совершающий никаких действий, то есть все эти коды не производят никакого эффекта. После кода NOP выполнение продолжается так, как если бы этого оператора вообще не было.

Таким образом, неустойчивым разветвлениям предоставляется возможность изменить семантику кода NOP, придавая ему новый смысл. Например, документ BIP-65 (`CHECKLOCKTIMEVERIFY`) определяет альтернативную интерпретацию кода NOP2. Клиенты с реализацией BIP-65 интерпретируют NOP2 как операцию `OP_CHECKLOCKTIMEVERIFY` и применяют правило консенсуса о безусловной фиксации времени для данных UTXO, которые содержат этот код операции в своем блокирующем скрипте. Это изменение является неустойчивым разветвлением, так как транзакция, корректная по условиям BIP-65, также остается корректной для любого клиента, который не воспользовался реализацией BIP-65 (или проигнорировал ее). Для старых клиентов скрипт содержит код операции NOP, который не производит никаких действий.

Другие способы обновления с использованием неустойчивых разветвлений

Альтернативная интерпретация кодов операций NOP была планируемым и вполне очевидным механизмом обновления правил консенсуса. Но не так

давно был введен другой механизм неустойчивых разветвлений, который не использует кодов NOP для особенного типа изменений правил консенсуса. Более подробно этот механизм описан в приложении Г. Segwit – это архитектурное изменение в структуре транзакции, которое перемещает разблокирующий скрипт (доказательство – WITness) из тела транзакции во внешнюю структуру данных (отделяет скрипт от транзакции – SEGregate). Изначально механизм Segwit рассматривался как обновление с устойчивым разветвлением, поскольку он изменяет основополагающую структуру (транзакцию). В ноябре 2015 года один из разработчиков Bitcoin Core предложил механизм, реализующий Segwit как неустойчивое разветвление. Этот механизм представлял собой изменение блокирующего скрипта для данных UTXO, созданных по правилам Segwit, но изменение выполнялось таким образом, что необновленные клиенты интерпретировали этот новый блокирующий скрипт как погашающий (redeemable) скрипт в комбинации с любым возможным разблокирующим скриптом. В результате механизм Segwit может быть введен в эксплуатацию без необходимости обновления каждого узла и без разделения основной цепочки, то есть это типичное неустойчивое разветвление.

Вероятно, существуют и другие, пока еще не открытые механизмы обновления с сохранением совместимости снизу вверх, функционирующие как неустойчивые разветвления.

Критика неустойчивых разветвлений

Неустойчивые разветвления, основанные на кодах операций NOP, не вызывают почти никаких возражений. Коды NOP были введены в язык Bitcoin Script именно с этой вполне определенной целью: обеспечение обновлений без нарушения нормального режима работы.

Тем не менее многие разработчики обеспокоены тем, что другие методы обновлений с использованием неустойчивых разветвлений приводят к неприемлемым компромиссам. Ниже перечислены основные пункты, по которым критикуются неустойчивые разветвления:

- унаследованные (из прошлого) технические ограничения (technical debt) – поскольку неустойчивые разветвления технически более сложны, чем обновления с использованием устойчивых разветвлений, они привносят унаследованные (из прошлого) технические ограничения (technical debt). Этот термин обозначает постоянное увеличение в будущем стоимости сопровождения кода из-за компромиссов в процессе проектирования ПО, принятых в прошлом (на начальной стадии создания ПО). В свою очередь, сложность кода увеличивает вероятность возникновения в нем ошибок и уязвимостей в подсистеме защиты;
- ослабление условий валидации – клиенты без обновления воспринимают транзакцию как корректную (валидную) без проверки измененных правил консенсуса. В действительности такие клиенты не выполняют проверку валидности с использованием полного набора правил консен-

суса, так как вообще не знают о новых правилах. Это относится не только к обновлениям на основе кодов операций NOP, но и ко всем прочим типам обновлений с использованием неустойчивых разветвлений;

- необратимость обновлений – поскольку неустойчивые разветвления создают транзакции с дополнительными ограничениями на основе правил консенсуса, обновления становятся практически необратимыми. Если обновление с использованием неустойчивого разветвления отменить после его активации, то все транзакции, созданные по новым правилам, станут некорректными по старым правилам, что приведет к безвозвратной потере денежных средств. Например, если CLTV-транзакция проверяется по старым правилам, то ограничения с блокировками по времени отсутствуют, и данные этой транзакции могут быть израсходованы в любое время. Таким образом, критики настаивают на том, что некорректное неустойчивое разветвление, которое должно быть отменено из-за программной ошибки, почти всегда будет приводить к потере денежных средств.

Оповещение о неустойчивом разветвлении с помощью поля версии блока

Поскольку неустойчивые разветвления позволяют клиентам без обновлений продолжать работу с механизмом консенсуса, механизм «активации» неустойчивых разветвлений предназначен для распространения среди майнеров оповещения о готовности: большинство майнеров обязательно должно подтвердить свою готовность и согласие на внедрение новых правил консенсуса. Для координации их действий существует механизм оповещения, позволяющий продемонстрировать поддержку изменений в правилах консенсуса. Этот механизм был введен вместе с принятием стандарта BIP-34 в марте 2014 года и заменен при вводе в действие стандарта BIP-9 в июле 2016 года.

Оповещение и активация по стандарту BIP-34

Первая реализация, определенная в стандарте BIP-34, использовала поле версии блока, чтобы позволить майнерам сообщить о готовности принять конкретное изменение в правилах консенсуса. До введения стандарта BIP-34 в поле версии блока записывалось значение 1 по общему соглашению, никоим образом не зависящему от правил консенсуса.

Стандарт BIP-34 определил изменение правила консенсуса, для которого требовалось поле coinbase (входные данные) coinbase-транзакции для размещения высоты блока. До BIP-34 поле coinbase могло содержать произвольные данные по усмотрению майнера. После ввода в действие стандарта BIP-34 корректные (валидные) блоки должны были содержать определенное значение высоты блока в начале записи coinbase и идентифицироваться по номеру версии, большему или равному 2.

Для оповещения об изменениях и активации стандарта BIP-34 майнеры устанавливают номер версии блока 2 вместо 1. Но это не сделало блоки версии 1 сразу же некорректными. После активации стандарта блоки версии 1 должны были стать некорректными, а все блоки версии 2 должны были обязательно содержать высоту блока в поле coinbase для признания их корректными.

Стандарт BIP-34 определил двухэтапный механизм активации, основанный на скользящем окне с размером в 1000 блоков. Майнер должен сообщать о своей готовности к принятию стандарта BIP-34, формируя блоки с явным указанием версии 2. Строго говоря, эти блоки пока еще не обязаны были обеспечивать полную совместимость с новым правилом консенсуса при включении высоты блока в coinbase-транзакцию, так как это новое правило консенсуса еще не было активировано. Активация правил консенсуса выполняется в два этапа:

- если 75% (750 из 1000 самых последних блоков) помечены номером версии 2, то блоки версии 2 обязательно должны содержать значение высоты блока в coinbase-транзакции, иначе они будут отброшены как некорректные. Блоки версии 1 продолжают приниматься сетью, но не обязаны содержать значение высоты блока. В этот период совместно действуют старые и новые правила консенсуса;
- когда 95% (950 из 1000 самых последних блоков) помечены номером версии 2, блоки версии 1 перестают считаться корректными. Блоки версии 2 являются корректными, только если они содержат значение высоты блока в поле coinbase (в соответствии с предыдущим пороговым значением). С этого времени все блоки должны обеспечивать полную совместимость с новыми правилами консенсуса и все корректные блоки непременно должны содержать значение высоты блока в coinbase-транзакции.

После успешного оповещения и активации по правилам стандарта BIP-34 этот механизм использовался еще два раза для активации неустойчивых разветвлений:

- стандарт BIP-66 Strict DER Encoding of Signatures был активирован по образу и подобию стандарта BIP-34 с оповещением о версии блока 3 и объявлением блоков версии 2 некорректными;
- стандарт BIP-65 `CHECKLOCKTIMEVERIFY` был активирован по образу и подобию стандарта BIP-34 с оповещением о версии блока 4 и объявлением блоков версии 3 некорректными.

После активации стандарта BIP-65 механизм оповещения и активации BIP-34 был отменен и заменен механизмом оповещения по стандарту BIP-9, описанному в следующем разделе.

Рассмотренный в этом разделе стандарт определен в документе BIP-34 Block v2, Height in Coinbase (https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki).

Оповещение и активация по стандарту BIP-9

Механизм, использованный в стандартах BIP-34, BIP-66 и BIP-65, был успешно применен при активации трех неустойчивых разветвлений. Тем не менее он был заменен из-за присущих ему следующих ограничений:

- использование целого числа в качестве номера версии блока, вследствие чего за один раз возможна только одна процедура активации неустойчивого разветвления, так как требуется приведение в полное соответствие предложений и окончательного соглашения по их приоритетам и последовательности введения;
- поскольку номер версии блока увеличивается на 1, механизм не мог обеспечить реализацию четко определенного способа отмены изменения и последующего предложения другого варианта изменения. Если продолжают работать старые клиенты, то они могут ошибаться при оповещении о новом изменении, которое в действительности представляет собой ранее отмененное изменение;
- каждое новое изменение необратимо сокращает диапазон доступных номеров версий блоков для будущих изменений.

Для устранения этих недостатков был предложен стандарт BIP-9, который, кроме того, улучшил эффективность и упростил реализацию внесения будущих изменений.

Стандарт BIP-9 интерпретирует номер версии блока как битовое поле, а не целое число. Так как изначально номер версии блока использовался как целое число от 1 до 4, только 29 битов остались доступными для работы с ними как с битовым полем. Таким образом, 29 битами можно воспользоваться для независимых и одновременных оповещений о готовности принять 29 различных предложений.

Стандарт BIP-9 также устанавливает максимальное время для оповещения и активации. Благодаря этому майнерам не нужно постоянно генерировать оповещения. Если предложение не активировано за интервал `TIMEOUT` (определенный в самом предложении), то такое предложение считается отвергнутым. Предложение может быть повторно выдвинуто для оповещения с установкой другого бита и с восстановлением периода активации.

Более того, по истечении интервала `TIMEOUT`, если предложение было активировано или отвергнуто, бит оповещения может быть повторно использован для очередного предложения без каких-либо затруднений. Таким образом, одновременно можно создавать до 29 оповещений об изменениях, а после завершения интервала `TIMEOUT` освободившиеся биты можно повторно использовать для предложения новых изменений.

Несмотря на то что биты оповещения можно повторно использовать или рециклировать даже до завершения периода голосования (подтверждения), авторы стандарта BIP-9 рекомендуют повторно использовать эти биты только при реальной необходимости. Дело в том, что из-за ошибок в более старом программном обеспечении может проявляться непредсказуемое поведение. Если говорить кратко, не следует обращаться к повторному использованию до тех пор, пока не будет полностью завершен первоначальный цикл использования всех 29 битов.

Предлагаемые изменения определяются структурой данных, содержащей следующие поля:

- name – краткое описание характерных отличительных признаков конкретного предложения. Чаще всего указывается документ BIP, описывающий предложение, в форме bipN, где N – номер документа BIP;
- bit – число от 0 до 28, номер бита в поле версии блока, в котором майнер размещает подтверждение принятия оповещения об этом предложении;
- starttime – время (основанное на Median Time Past, MTP) начала действия оповещения, отсчитываемое с момента интерпретации значения бита как оповещения о готовности принять данное предложение;
- endtime – время (основанное на MTP), по истечении которого изменение считается отвергнутым, если не было достигнуто пороговое значение активации.

В отличие от BIP-34, стандарт BIP-9 считает периоды оповещения об активации в целых интервалах на основе периода изменения целевого значения и уровня сложности 2016 блоков. В каждом интервале изменения целевого значения, если общее количество блоков с оповещением о принятии предложения превышает 95% (1916 блоков из 2016), такое предложение будет активировано в следующем интервале изменения целевого значения.

В стандарте BIP-9 приводится диаграмма состояний предложения для наглядного представления состояний и переходов между ними для отдельного предложения, как показано на рис. 10.10.

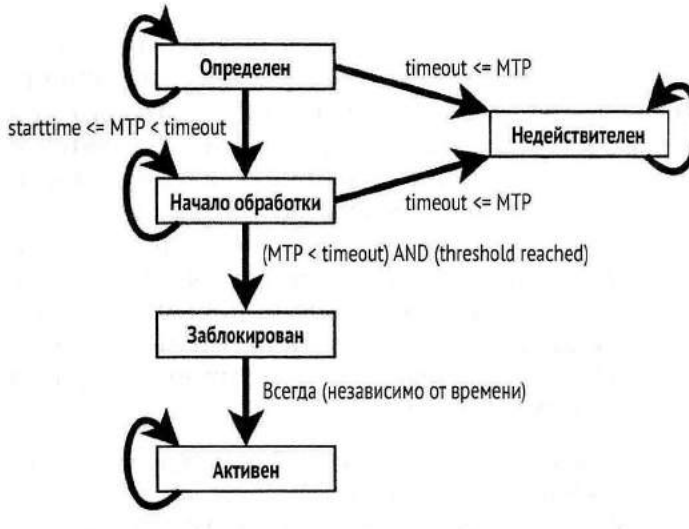


Рис. 10.10 ❖ Диаграмма состояний предложений и переходов между состояниями по стандарту BIP-9

Сначала предложение находится в состоянии `DEFINED`, сразу после того, как его параметры известны (определены) в программном обеспечении биткойна. Для блоков с MTP, прошедшим после времени начала, состояние предложе-

ния изменяется на STARTED. Если пороговое значение голосования превышено в течение периода изменения целевого значения, а тайм-аут не был исчерпан, то предложение переходит в состояние LOCKED_IN. В следующем периоде изменения целевого значения это предложение становится активным ACTIVE. Предложения остаются в состоянии ACTIVE постоянно после достижения этого состояния. Если тайм-аут истекает ранее достижения порогового значения голосования, то состояние предложения изменяется на FAILED, то есть такое предложение отвергается. Отвергнутые (REJECTED) предложения остаются в этом состоянии постоянно.

Стандарт BIP-9 впервые был реализован для активации функции CHECKSEQUENCEVERIFY и соответствующих стандартов BIP-68, BIP-112, BIP-113. Это предложение, названное «csv», было успешно активировано в июле 2016 года.

Рассмотренный в этом разделе стандарт определен в документе BIP-9 Version bits with timeout and delay (https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki).

Разработка программного обеспечения для механизма консенсуса

Разработка программного обеспечения для механизма консенсуса продолжается, кроме того, активно обсуждаются различные механизмы внесения изменений в правила консенсуса. Благодаря своей коренной сущности биткойн устанавливает весьма высокий уровень требований к координации и согласованию изменений. Являясь децентрализованной системой, биткойн-сеть не предполагает присутствия какого-либо «органа управления», навязывающего свою волю членам сети. Все ресурсы и мощности распределены между многочисленными компонентами сети, такими как майнеры, разработчики ядра, разработчики кошельков, пункты обмена, торговые площадки и конечные пользователи. Решения не могут приниматься в одностороннем порядке любым из этих компонентов. Например, несмотря на то что майнеры теоретически могут изменять правила простым большинством (51%), они ограничены тем, что другие компоненты сети имеют полное право выражать свое согласие или несогласие с этими изменениями. Если майнеры действуют в одностороннем порядке, то остальные члены сети могут просто отказаться следовать за ними, поддерживая экономическую деятельность в цепочке меньшинства. При отсутствии экономической активности (транзакции, торговые сделки, кошельки, обменные и конвертационные операции) майнеры будут генерировать не имеющие никакой ценности денежные единицы в пустых блоках. Такое распыление ресурсов и мощностей приводит к ситуации, в которой либо все участники непременно должны согласовывать свои действия, либо внесение изменений становится невозможным. Текущим состоянием является стабильное состояние биткойн-системы с возможностью внесения лишь небольшого количества изменений при условии достижения убедительного

консенсуса, поддерживаемого подавляющим большинством голосов. Пороговое значение 95% для принятия неустойчивых разветвлений отражает это реальное состояние.

Очень важно осознать, что не существует идеального решения для разработки и развития механизма консенсуса. Необходимы компромиссы между введением устойчивых и неустойчивых разветвлений. Для некоторых типов изменений наилучшим выбором могут стать неустойчивые разветвления, в других случаях лучше применять устойчивые разветвления. Однозначно правильного варианта выбора нет, оба связаны с определенными рисками. Единственной постоянной характеристикой процесса разработки программного обеспечения для механизма консенсуса является тот факт, что любое изменение затруднительно и достижение консенсуса вынуждает идти на определенные компромиссы.

Некоторые специалисты видят в этом слабость систем с механизмом консенсуса. Возможно, со временем вы придете к той же точке зрения, которой придерживается автор книги: механизм консенсуса является самой сильной стороной биткойн-системы.

Глава 11

Обеспечение безопасности биткойн-системы

Защита биткойн-системы представляет собой весьма трудную задачу, так как сам по себе биткойн не является абстрактной ссылкой на значение, подобно балансу в банковском счете. Биткойн гораздо больше похож на цифровые деньги или золото. Вероятно, вы слышали такое выражение: «Собственность – это на девять десятых закон» или «Собственность диктует законы, владелец на девять десятых прав» («Possession is nine-tenths of the law»). Разумеется, в биткойн-системе собственность – это на десять десятых закон. Владение ключами для разблокировки биткойнов равнозначно владению наличными деньгами или слитком драгоценного металла. Вы можете потерять биткойны, они могут «куда-то подеваться», их могут украсть, или вы по неосторожности можете отдать кому-то неверную сумму. В каждом из этих случаев пользователи оказываются в таком же положении, как если бы они случайно уронили наличные деньги на многолюдной улице.

Тем не менее биткойн обладает особыми свойствами, которых нет у наличных денег, золота и банковского счета. Для биткойн-кошелька, содержащего ваши ключи, можно создавать резервные копии, как для любого обычного файла. Биткойн-кошелек может храниться в виде нескольких копий, его можно даже распечатать на бумаге для создания «твердой» (вещественной) резервной копии. А вот для наличных денег, золота или банковского счета создать «резервную копию» вы не сможете. Биткойн заметно отличается от всевозможных средств платежа, существовавших до него, поэтому и обеспечение безопасности биткойна мы должны рассматривать как новый, ранее не встречавшийся образ действий.

Основы обеспечения безопасности

Основным принципом работы биткойн-системы является децентрализация, и этот принцип очень важен для обеспечения защиты. Централизованная модель, такая как обычная банковская или платежная сеть, зависит от подсисте-

мы управления доступом и тщательной проверки, не позволяющей злоумышленникам войти в систему. Децентрализованная система, подобная биткойну, напротив, передает всю ответственность и управление в руки пользователей. Поскольку защита биткойн-сети основана на алгоритме доказательства выполнения работы (Proof-of-Work, PoW), управление доступом как таковое отсутствует, сеть может быть всегда открытой и не требует шифрования биткойн-трафика.

В обычной платежной сети, например в системе кредитных карт, сам платеж не ограничен по времени и не связан какими-либо условиями, так как платежные данные содержат личный идентификатор пользователя (номер кредитной карты). После первоначальной платежной операции все, кому известен этот идентификатор, могут «ощипывать» денежные средства и все время записывать расходы на счет владельца. Поэтому платежная сеть должна быть полностью защищена с помощью средств шифрования и должна гарантировать, что никакие «прослушиватели» и «перехватчики» не смогут скомпрометировать платежный трафик во время передачи или после его сохранения (в резервных копиях). Если злоумышленник получает доступ к такой системе, он может вносить опасные изменения в текущие транзакции и отдельные элементы платежных данных, которые могут использоваться для создания новых транзакций. Усугубляет ситуацию то, что при компрометации данных одного клиента все клиенты подвержены риску похищения их идентификационных данных, поэтому неизбежно возникает необходимость принятия мер по предотвращению мошеннического использования скомпрометированных учетных записей (счетов).

Биткойн-система в корне отличается от описанной выше платежной системы. Биткойн-транзакция авторизуется с помощью единственного специализированного значения, адресуется конкретно указанному получателю и не может быть подделана или изменена. В транзакции не содержится никакой секретной информации, такой как идентификационные характеристики членов сети, следовательно, ее невозможно использовать для авторизации дополнительных платежей. Таким образом, платежная биткойн-сеть не нуждается в шифровании или в специальной защите от «подслушивающих». Вы можете свободно распространять биткойн-транзакции по общедоступным каналам связи, таким как незащищенный Wi-Fi или Bluetooth, без какого-либо ущерба для безопасности.

Децентрализованная модель защиты биткойн-системы передает почти всю власть в руки пользователей. Но вместе с властью передается ответственность за обеспечение секретности ключей. Большинству пользователей не так-то просто это сделать, особенно на устройствах общего назначения, таких как смартфоны и ноутбуки, подключенные к Интернету. Несмотря на то что децентрализованная модель биткойн-системы способна предотвратить тот тип массовой компрометации учетных записей, который присущ системе с использованием кредитных карт, многие пользователи просто не способны защитить свои ключи надлежащим образом, поэтому их взламывают, одного за другим.

Разработка защищенных биткойн-систем

Самым важным основополагающим принципом для разработчиков биткойна является децентрализация. Большинство разработчиков хорошо знакомо с централизованными моделями защиты, поэтому склонно применять эти модели и в своих биткойн-приложениях, но результаты оказываются катастрофическими.

Подсистема защиты биткойн-сети основана на децентрализованном управлении ключами и на независимой проверке (валидации) транзакций майнерами. Если требуется повышение уровня защиты биткойн-системы, необходимо проследить за тем, чтобы не выйти за пределы модели защиты биткойна. Проще говоря, не отнимайте управления ключами у пользователей и не создавайте транзакций вне структуры данных блокчейна.

Например, на начальном этапе существования биткойн-системы многие расчетные пункты объединяли все денежные средства пользователей в одном «горячем» кошельке с хранением ключей на единственном сервере. Такое решение лишает пользователей возможности управления и создает централизованную систему управления всеми ключами. Многие подобные системы были взломаны с губительными последствиями для их клиентов.

Еще одна широко распространенная ошибка – вынос транзакций «за пределы структуры данных блокчейна» в бесплодных попытках снижения суммы отчислений за транзакции или ускорения обработки транзакций. Любая система «за пределами структуры данных блокчейна» фиксирует транзакции в своем внутреннем централизованном реестре и только изредка будет синхронизировать эти записи со структурой данных блокчейна биткойн-системы. Но и этот практический подход также заменяет децентрализованную модель защиты биткойна собственной централизованной моделью. Когда транзакции располагаются вне структуры данных блокчейна, недостаточно защищенные централизованные реестры можно совершенно незаметно подделать, вывести из них денежные средства и опустошить денежный запас.

Если вы не готовы к огромным инвестициям в обеспечение защиты при эксплуатации системы, в организацию многоуровневого управления доступом и аудита (как это сделано в обычных банках), то следует крепко поразмыслить, прежде чем выводить денежные средства из-под децентрализованного механизма защиты биткойн-системы. Даже если вы располагаете средствами и возможностями для реализации надежной модели защиты, такое проектное решение всего лишь воспроизводит уязвимую модель обычных финансовых сетей, подверженную рискам похищения идентификационных данных, повреждения и присвоения чужих денег. Чтобы воспользоваться всеми преимуществами (пока) единственной в своем роде децентрализованной модели защиты биткойна, необходимо всячески избегать соблазна применить централизованные архитектурные модели, которые могут казаться более привычными, но в корне искажают основы системы защиты биткойна.

Основа доверительных отношений

Обычная архитектурная модель защиты базируется на концепции, называемой «основой доверительных отношений» (root of trust), то есть заслуживающее доверия ядро, используемое как основание для защиты всей системы или приложений. Архитектура подсистемы защиты разработана в виде последовательности концентрических сфер, окружающих это «ядро доверительных отношений», подобно слоям луковицы, распространяя доверительные отношения от центра к периферии. Каждый уровень создается на основе более надежного в плане доверия внутреннего уровня с использованием средств управления доступом, цифровых подписей, шифрования и прочих элементов защиты. Когда программные системы становятся более сложными, в них возрастает вероятность возникновения ошибок, из-за которых система становится более уязвимой с точки зрения защиты. Очевидно, что чем более сложной становится программная система, тем труднее ее защитить. Концепция основы доверительных отношений гарантирует, что поддержка бóльшей части доверительных отношений размещается в наименее сложных внутренних частях системы, следовательно, менее уязвима, тогда как более сложное программное обеспечение вынесено в наружные слои, окружающие ядро. Эта схема архитектуры защиты повторяется в различных масштабах, сначала формируя ядро доверительных отношений на аппаратном уровне отдельной системы, затем расширяет сферу действия этого ядра через операционную систему на более высокий уровень системных сервисов, наконец, выходит на уровень многочисленных серверов, расположенных в концентрических сферах с убыванием степени доверия.

Но архитектура защиты биткойн-системы отличается от описанной выше схемы. В биткойн-сети система консенсуса создает надежный в плане доверия общедоступный реестр, который полностью децентрализован. Проверенная корректная структура данных блокчейна использует первичный блок как основу доверительных отношений, создавая «цепочку доверия» поверх текущего блока. Биткойн-системы могут и должны использовать структуру данных блокчейна в качестве собственной основы доверительных отношений. При проектировании сложного биткойн-приложения, состоящего из сервисов многих различных систем, необходимо тщательно продумать архитектуру подсистемы защиты, чтобы точно определить место размещения ядра доверительных отношений. В конечном счете единственной сущностью, которой следует полностью доверять, является проверенная корректная структура данных блокчейна. Если приложение явно или косвенно переносит поддержку доверительных отношений в объект, отличный от структуры данных блокчейна, то такому объекту необходимо уделить особое внимание, так как он становится источником потенциальных уязвимостей. Надежный метод проектирования архитектуры подсистемы защиты приложения состоит в том, что каждый его компонент рассматривается в отдельности и исследуются предполагаемые сценарии, в которых этот компонент полностью скомпрометирован и нахо-

дится под управлением злоумышленника. Необходимо исследовать каждый компонент приложения поочередно и оценивать степень воздействия на всю подсистему защиты в целом, если компонент будет скомпрометирован. Если при компрометации каких-либо компонентов приложение становится незащищенным, это свидетельствует об ошибочном размещении поддержки доверительных отношений в таких компонентах. Биткойн-приложение без собственных уязвимостей должно быть уязвимым только при нарушении работы механизма консенсуса биткойна. Это означает, что основа доверительных отношений приложения базируется на самой надежной и прочной части архитектуры подсистемы защиты биткойн-системы.

Многочисленные примеры взлома отдельных операционных узлов биткойн-сети подкрепляют эту точку зрения, так как архитектура подсистемы защиты и проектные решения этих узлов не выдерживают никакой критики даже при самом поверхностном исследовании. Их централизованные реализации в явном виде переносили поддержку доверительных отношений в многочисленные компоненты за пределами структуры данных блокчейна биткойн-системы, например в «горячие» кошельки, централизованные базы данных реестра, ненадежные ключи шифрования и прочие подобные объекты.

Наиболее эффективные практические методики защиты пользователей

Люди использовали физические средства управления защитой на протяжении тысячелетий. Область знаний, ведающая цифровыми системами защиты, появилась менее 50 лет назад. Современные операционные системы общего назначения недостаточно защищены и не предназначены специально для хранения цифровых денег. Наши компьютеры постоянно находятся под угрозой, исходящей извне, так как постоянно подключены к Интернету. На компьютерах работают тысячи программных компонентов от сотен авторов, зачастую с неограниченным доступом к файлам пользователя. Единственный фрагмент мошеннического программного обеспечения, затесавшийся среди тысяч прочих компонентов, установленных на вашем компьютере, способен нарушить работу клавиатуры, испортить файлы, украсть биткойны, хранящиеся в приложениях кошельков. Уровень обслуживания компьютеров, необходимый для надежной защиты от вирусов и троянских программ, гораздо выше, чем реальный уровень подготовки практически всех пользователей, за исключением чрезвычайно малой их части, обладающей требуемыми техническими навыками и знаниями.

Несмотря на десятилетия исследований и достижений в области информационной безопасности, цифровые активы (ценности) остаются удручающе уязвимыми для определенной части злоумышленников. Даже самые защищенные системы с многочисленными ограничениями доступа в финансовых компаниях, спецслужбах и предприятиях оборонной промышленности достаточно час-

то взламываются. Биткойн создает цифровые активы, обладающие собственной, присущей только им ценностью, поэтому может быть похищен и передан новым владельцам мгновенно и безвозвратно. Это является вожделенной целью для взломщиков. Раньше взломщикам приходилось преобразовывать идентификационную информацию или элементы учетной записи – такие как номера кредитных карт или номера банковских счетов – в материальные ценности после взлома и несанкционированного доступа к приватным данным. Несмотря на трудности, связанные с сокрытием украденного и «отмыванием» капиталов и прочих активов, наблюдается постоянный рост количества преступлений в финансовой сфере. Биткойн усугубляет эту проблему, поскольку не требует сокрытия и «отмывания», он сам по себе является ценностью, выражаемой в форме цифрового имущества.

К счастью, биткойн также создает стимулы к усовершенствованию информационной защиты. Раньше опасность несанкционированных действий на компьютерах была не вполне определенной и не оказывала прямого воздействия, но с появлением биткойна эти риски стали ясными и очевидными. Поддержка биткойн-системы на компьютерах сосредоточила внимание пользователей на необходимости усовершенствования защиты компьютеров. Прямым следствием быстрого и все более широкого распространения биткойна и других цифровых валют стало явно прослеживаемое развитие как практических приемов взлома, так и решений по компьютерной защите. Проще говоря, теперь у взломщиков появилась весьма привлекательная цель, а у пользователей – очевидный стимул для защиты себя и своей собственности.

За последние три года самым ярким последствием широкого распространения и применения биткойна стал огромный объем инноваций в области информационной защиты, проявленный в форме средств аппаратного шифрования, аппаратных хранилищ ключей и аппаратных кошельков, технологии мультиподписей и цифрового депонирования у третьего лица. В следующих разделах мы более подробно рассмотрим некоторые наиболее эффективные практические методики защиты пользователей.

Физические средства хранения биткойнов

Поскольку большинство пользователей чувствует себя намного увереннее при обеспечении физической защиты, по сравнению с обеспечением информационной (цифровой) защиты, весьма эффективной методикой защиты биткойнов является преобразование их в некоторую физическую форму. Биткойн-ключи представляют собой не что иное, как длинные числа. Это означает, что их можно хранить в физической форме, например в виде распечаток на бумаге или в виде гравировки на металлической монете. Защита ключей становится таким же простым делом, как физическая защита печатной копии биткойн-ключей. Набор биткойн-ключей, распечатанных на бумаге, называют «бумажным кошельком», и для его создания существует множество бесплатных инструментальных средств. Автор хранит большую часть своих биткойнов (99%

или даже больше) в бумажных кошельках, зашифрованных по стандарту BIP-38, с многочисленными резервными копиями, размещенными в безопасных местах. Хранение биткойнов в режиме офлайн называется «холодным хранением» (cold storage), которое является одной из самых надежных методик защиты. В системе холодного хранения ключи генерируются на офлайновой системе (никогда не подключаемой к Интернету) и сохраняются также в режиме офлайн либо на бумаге, либо на цифровом носителе, например на устройстве памяти USB.

Аппаратные кошельки

В конечном итоге система защиты биткойнов постепенно должна принять форму аппаратных кошельков, защищенных от воровства и неумелого обращения. В отличие от смартфона или настольного компьютера, аппаратный кошелек предназначен для единственной цели – безопасного хранения биткойнов. При отсутствии уязвимого программного обеспечения общего назначения и с ограниченными средствами интерфейса аппаратные кошельки способны обеспечить почти максимальный уровень защиты от неосторожного обращения и ошибочных действий слабо подготовленных пользователей. Автор предполагает, что аппаратные кошельки уже сейчас становятся преобладающим способом хранения биткойнов. Хорошим примером аппаратного кошелька является устройство Trezor (https://trezor.io/).

Разумный баланс защиты и рисков

Почти всем пользователям хорошо известны факты похищения биткойнов, но здесь кроется еще бóльшая опасность. Файлы данных теряются постоянно. Если эти файлы содержат биткойны, то потери намного более критичны. Прилагая усилия по защите своих биткойн-кошельков, пользователи должны быть очень внимательными, чтобы не перестараться и не потерять биткойны. В июле 2011 года произошел широкоизвестный инцидент, в результате которого образовательный проект потерял почти 7000 биткойнов. Стараясь предотвратить хищения, владельцы реализовали сложную последовательность зашифрованных резервных копий. В конце концов, они оказались в полном проигрыше, когда случайно потеряли ключи шифрования и все резервные копии стали бесполезными. Если вы слишком хорошо защищаете свои биткойны, то может возникнуть ситуация, когда вы не сможете их извлечь. Здесь уместна аналогия с реальными деньгами, настолько хорошо запрятанными где-то в недрах пустыни, что потом их почти невозможно найти даже владельцу.

Диверсификация рисков

Согласились бы вы хранить весь свой капитал в виде наличных денежных знаков в кошельке? Большинство людей считало бы такое решение опрометчивым, но все же пользователи часто хранят все свои биткойны в одном ко-

шельке. Вместо этого пользователям следовало бы распределять вероятность возникновения рисков по нескольким различным биткойн-кошелькам. Предусмотрительные пользователи будут хранить лишь небольшую часть, возможно, менее 5% своих биткойнов в онлайн-режиме или в мобильном кошельке как «карманные деньги на мелкие расходы». Остальные биткойны должны быть поделены между несколькими разнообразными механизмами хранения, такими как кошелек на настольном компьютере и офлайновый кошелек (холодное хранение).

Мультиподпись и управление

Если компания или частное лицо хранит весьма большую сумму в биткойнах, следует рассмотреть вариант с использованием биткойн-адреса с функцией мультиподписи. Такие адреса обеспечивают защиту денежных средств, требуя более одной подписи для совершения платежа. Ключи для подписи должны храниться в нескольких различных местах и управляться различными людьми. Например, в корпоративной среде ключи должны генерироваться независимо друг от друга и храниться у нескольких руководящих работников компании. При таком подходе один человек не сможет нанести какого-либо ущерба денежным фондам. Кроме того, адреса с функцией мультиподписи могут обеспечить определенный уровень избыточности, при котором один человек является владельцем нескольких ключей, хранящихся в различных местах.

Жизнеспособность

Одним важным условием обеспечения безопасности, которое часто недооценивают, является доступность, особенно в ситуациях полной недееспособности или смерти владельца ключей. Пользователи биткойн-системы предупреждены о необходимости применения сложных паролей и хранения своих ключей с обеспечением защиты и секретности и с абсолютным исключением возможности их предъявления любому постороннему лицу. К сожалению, такой подход делает практически невозможным обращение к денежным средствам членов семьи, если сам пользователь по каким-либо причинам не может разблокировать свои денежные средства. А в большинстве случаев семьям пользователей биткойн-системы вообще ничего неизвестно о существовании денежных сумм в биткойнах.

Если вы являетесь владельцем большого количества биткойнов, то вам следует серьезно подумать о нюансах обеспечения совместного доступа для своих родственников или адвоката, которым вы доверяете. Можно создать более сложную схему обеспечения жизнеспособности с предоставлением доступа только по мультиподписи и предварительного планирования распоряжения имуществом адвокатом, специализирующимся в области «душеприказчик-распорядитель цифровым имуществом».

Резюме

Биткойн – это совершенно новая, беспрецедентная и сложная технология. Со временем будут разработаны более надежные средства защиты и практические методики, упрощающие ее применение неподготовленными пользователями. В настоящее время пользователи биткойн-системы могут воспользоваться многими советами и рекомендациями, описанными здесь, чтобы без проблем и излишних рисков работать со своими биткойнами.

Глава 12

Приложения блокчейна

Рассмотрим биткойн как прикладную платформу (application platform), чтобы лучше понять его сущность. Сейчас многие употребляют термин «блокчейн» для обозначения любой прикладной платформы, использующей основные принципы биткойна. Этот термин часто применяют неправильно, обозначая им многие объекты, не обладающие основными свойствами, присущими структуре данных блокчейна биткойн-системы.

В этой главе мы будем изучать функциональные свойства, предлагаемые структурой данных блокчейна биткойн-системы как прикладной платформы. Мы рассмотрим базовые элементы (primitives), представляющие собой основные конструкции-компоненты для создания любого приложения блокчейна. Кроме того, будут описаны некоторые важные приложения, использующие эти базовые элементы, такие как цветные монеты, каналы платежей (каналы состояний) и каналы платежей с маршрутизацией (Lightning Network).

Введение

Биткойн-система была задумана и спроектирована как децентрализованная финансовая и платежная система. Но большая часть ее функциональности унаследована от многих конструкций более низкого уровня, которые могут использоваться для разнообразных приложений более широкого профиля. При создании биткойн-системы не применялись такие компоненты, как счета, пользователи, балансы и платежи. Вместо этого использовался специализированный скриптовый язык для описания транзакций с криптографическими функциями низкого уровня, который мы рассматривали в главе 6. Из упомянутых выше базовых элементов можно формировать концепции более высоких уровней – те же счета, балансы и платежи, но этим дело не ограничивается: возможно создание множества других сложных приложений. Таким образом, структура блокчейна биткойн-системы вполне может стать прикладной платформой, предлагающей надежные сервисы для приложений, такие как смарт-контракты, значительно превосходящие изначальные возможности и цели цифровой валюты и платежей.

Базовые элементы

При правильно организованной работе в течение длительного времени биткойн-система обеспечивает определенные гарантии, которые могут использоваться как базовые элементы для создания приложений. Ниже приводится краткое описание этих базовых элементов:

- исключение двойного расходования – самая главная гарантия, обеспечиваемая алгоритмом децентрализованного консенсуса биткойн-системы, – полная уверенность в том, что данные UTXO не могут быть израсходованы дважды;
- неизменяемость – после добавления транзакции в структуру данных блокчейна и достаточного объема выполненной работы по добавлению последующих блоков данные транзакции становятся практически неизменяемыми. Неизменяемость обеспечивается энергозатратами, так как перезапись структуры данных блокчейна требует расхода электроэнергии для проведения доказательства выполнения работы (PoW). Затраты электроэнергии неизбежны, следовательно, степень неизменяемости возрастает с увеличением объема работы, выполненной для добавления блоков поверх блока, содержащего эту транзакцию;
- беспристрастность – децентрализованная биткойн-сеть распространяет корректные транзакции вне зависимости от источника или содержимого этих транзакций. Это означает, что каждый может создать корректную транзакцию с достаточными суммами отчислений и быть вполне уверенным в том, что эта транзакция будет передана в сеть и включена в структуру данных блокчейна;
- защищенная маркировка времени – правила консенсуса отвергают любой блок, метка времени которого слишком ранняя или слишком поздняя. Это гарантирует надежность меток времени в блоках. Метка времени в блоке обеспечивает защиту от преждевременного расходования входных данных всех транзакций, включенных в конкретный блок;
- авторизация – цифровые подписи, проверяемые в децентрализованной сети, обеспечивают авторизацию. Скрипты, содержащие требования к цифровой подписи, не могут быть выполнены без авторизации от имени владельца секретного ключа, на который косвенно ссылается конкретный скрипт;
- возможность проведения аудита – все транзакции общедоступны, их можно проверять в любое время. Все транзакции и блоки можно проследить в обратном порядке в непрерывной цепочке вплоть до первичного блока;
- учет денежных средств – в любой транзакции (за исключением coinbase-транзакции) сумма входных данных равна сумме выходных данных плюс отчисления за транзакции. Нет никакой возможности создать или уничтожить какую-либо сумму в биткойнах в транзакции. Сумма выходных данных не может превышать сумму входных данных;

- неограниченный срок хранения – срок хранения корректной транзакции никогда не истекает. Если транзакция корректна сегодня, она останется корректной в ближайшем будущем, пока ее входные данные остаются неизрасходованными и пока не изменятся правила консенсуса;
- целостность – биткойн-транзакции, подписанные SIGHASH_ALL, или части транзакции, подписанные другим типом SIGHASH, не могут быть изменены без нарушения корректности цифровой подписи, следовательно, без нарушения корректности всей транзакции;
- атомарность (неделимость) транзакции – биткойн-транзакции неделимы (атомарны), то есть либо корректны (валидны) и подтверждены (прошли процедуру майнинга), либо нет. «Частичные» транзакции не могут быть включены в процесс майнинга, для транзакций невозможно какое-либо промежуточное состояние. В любой момент времени транзакция или «отмайнена», или нет;
- дискретность (неделимость) денежных сумм – выходные данные транзакций дискретны, а денежные суммы, указанные в них, неделимы. Они могут быть израсходованы или неизрасходованы в полном объеме. Их нельзя разделить или израсходовать частично;
- возможность установления кворума для управления денежными средствами – ограничения, налагаемые механизмом мультиподписей в скриптах, формируют кворум для авторизации, предварительно определенный в схеме мультиподписей. Требование M-of-N (M-из-N) вводится правилами консенсуса;
- блокировка по времени/отложенное выполнение – любой элемент скрипта, содержащий относительную или абсолютную блокировку по времени, может быть выполнен только по истечении заданного интервала времени;
- репликация – децентрализованное хранение структуры блокчейна гарантирует, что транзакция после майнинга и после достаточного количества подтверждений многократно копируется в сети и становится надежной и устойчивой к случайным потерям данных, сбоям электропитания и т. п.;
- защита от подделок – транзакция может расходовать только реально существующие, проверенные, корректные выходные данные. Невозможно создать или подделать какую-либо денежную сумму;
- согласованность (непротиворечивость) – без какого-либо участия майнеров блоки, записанные в структуру данных блокчейна, подвергаются реорганизации, пересматривается их статус с экспоненциально уменьшающейся вероятностью, основанной на глубине расположения записанного блока. После того как глубина расположения блока становится значительной, для его изменения требуются существенные вычислительные и энергетические затраты, что делает внесение изменений практически невозможным;

- фиксация внешнего состояния – транзакция может передавать некоторое значение через код OP_RETURN, представляющий переход в определенное состояние в машине внешних состояний;
- предопределенный объем выпуска биткойнов – будет выпущено менее 21 миллиона биткойнов по предварительно определенному графику.

Приведенный список базовых элементов не является полным и может расширяться с появлением новых функциональных возможностей в биткойн-системе.

Приложения, создаваемые из базовых элементов

Базовые элементы, предлагаемые биткойном, представляют собой элементы надежной платформы, которую можно использовать для создания приложений. Ниже перечислены некоторые примеры таких приложений, которые существуют в настоящее время, а также используемые ими базовые элементы:

- Proof-of-Existence (Digital Notary) – доказательство существования (цифровой нотариальный сервис) – неизменяемость + метка времени + долговечность. Цифровой отпечаток может быть записан с помощью транзакции в структуру данных блокчейна, доказывая тем самым, что некий документ существовал (метка времени) в тот момент, когда он был зафиксирован. Такой цифровой отпечаток не может быть изменен задним числом (неизменяемость), а соответствующее доказательство будет храниться постоянно (долговечность);
- Kickstarter (Lighthouse) – согласованность (непротиворечивость) + атомарность (неделимость) + целостность – если вы подписываете один фрагмент входных данных и выходные данные (целостность) транзакции по сбору средств, другие лица могут присоединиться к кампании по сбору средств (сделать свой взнос), но израсходовать собранные средства невозможно (атомарность) до тех пор, пока не будет достигнута конечная цель кампании (то есть пока не будет собрана сумма, указанная в выходных данных) (согласованность);
- payment channels – каналы платежей – установление кворума для управления денежными средствами + блокировка по времени + исключение двойного расходования + неограниченный срок хранения + защита от цензуры + авторизация – мультиподпись по схеме 2-of-2 (кворум для управления) в сочетании с блокировкой по времени, используемая как «расчетная» (по соглашению) транзакция канала платежа, может быть сохранена (неограниченный срок хранения) и израсходована в любое время (защита от цензуры) определенной группой лиц (авторизация). Затем две группы лиц могут создать транзакции-обязательства, которые выполняют двойное расходование (исключение двойного расходования) расчетной транзакции с укороченной блокировкой по времени (блокировка по времени).

Цветные монеты

Первое приложение на основе блокчейна, которое мы будем рассматривать, называется цветные монеты (colored coins).

Сам термин «цветные монеты» обозначает набор взаимосвязанных технологий, который позволяет использовать биткойн-транзакции для записи операций создания, установления и передачи прав владения «внешним» имуществом (ресурсами и т. п.), отличным от биткойнов. Под «внешним» подразумевается имущество, которое не хранится непосредственно в структуре данных блокчейна биткойн-системы, в отличие от самих биткойнов, которые являются «внутренним» имуществом по отношению к структуре данных блокчейна.

Цветные монеты используются для прослеживания и управления цифровым имуществом, а также физическим имуществом, владельцами которого являются третьи стороны. Имуществом можно торговать, применяя для этого цветные монеты как сертификаты на право владения. Цветные монеты для цифрового имущества могут представлять нематериальные активы, такие как стокноты (торговые или биржевые сертификаты), лицензии, виртуальная собственность (элементы игр) или большинство любых форм защищаемой лицензиями интеллектуальной собственности (торговые марки, авторские права и т. п.). Цветные монеты для материальных ценностей могут представлять сертификаты на право владения товарами (золото, серебро, нефть), земельными участками, автомобилями, судами, летательными аппаратами и т. д.

Термин произошел от идеи «раскрашивать» или маркировать номинальные стоимости биткойнов, например один сатоши, чтобы обозначать значения, отличающиеся от одного биткойна. В качестве аналогии рассмотрим воображаемую маркировку банкноты 1 доллар США, включающую сообщение «this is a stock certificate of ACME» (это сертификат/акция ACME) или «this note can be redeemed for 1 oz of silver» (эта банкнота дает право на покупку 1 унции серебра), и продажу однодолларовой банкноты как сертификата на право владения другим указанным активом. Первая реализация цветных монет под названием Enhanced Padded-Order-Based Coloring, или EPOBC, устанавливала соответствие между «внешним» имуществом и выходными данными в сумме 1 сатоши. Это были действительно «цветные монеты», так как каждый тип имущества добавлялся как атрибут (цвет) одного сатоши.

Более поздние реализации цветных монет использовали скриптовый код операции `OP_RETURN` для хранения метаданных в транзакции в сочетании с внешними хранилищами данных, которые связывали эти метаданные с определенными типами имущества.

Двумя наиболее известными реализациями цветных монет на сегодняшний день являются Open Assets (http://www.openassets.org/) и Colored Coins by Colu (http://coloredcoins.org). Эти две системы применяют различные подходы к реализации цветных монет, поэтому не совместимы между собой. Цветные монеты, созданные в одной системе, невозможно увидеть или как-либо использовать в другой системе.

Использование цветных монет

Цветные монеты создаются, перемещаются и контролируются в специальных кошельках, которые могут интерпретировать метаданные протокола цветных монет, прикрепленные к биткойн-транзакциям. Необходимо предпринять особые меры, чтобы предотвратить использование ключей, связанных с цветными монетами, в обычных биткойн-кошельках, поскольку обычный кошелек может случайно уничтожить метаданные. Кроме того, цветные монеты не должны пересылаться по адресам, управляемым обычными кошельками, а только по адресам, которые управляются специальными кошельками, ориентированными на работу с цветными монетами. Обе системы – Colu и Open Assets – используют специальные адреса для цветных монет для снижения риска потери метаданных, гарантируя, что цветные монеты не будут отправлены в не предназначенные для них кошельки.

К тому же цветные монеты невозможно увидеть в большинстве проводников блокчейна общего назначения. Вместо них необходимо воспользоваться специализированным проводником-обозревателем цветных монет для правильной интерпретации метаданных соответствующих транзакций.

Приложение-кошелек и проводник блокчейна, совместимые с Open Assets, можно найти на сайте coinprism (https://www.coinprism.info).

Приложение-кошелек и проводник блокчейна, совместимые с Colu Colored Coins, можно найти на сайте Blockchain Explorer (http://coloredcoins.org/explorer/).

Подключаемый модуль Copay размещен на сайте Colored Coins Copay Addon (http://coloredcoins.org/colored-coins-copay-addon/).

Выпуск цветных монет

Каждая реализация применяет свой, отличный от других способ создания цветных монет, но все реализации предоставляют практически одинаковую функциональность. Процесс создания цветных монет, связанных с некоторым типом имущества, называется эмиссией (issuance). Первоначальная транзакция – эмиссионная транзакция (issuance transaction) – регистрирует конкретный элемент имущества в структуре данных блокчейна и создает идентификатор имущества (asset ID), используемый для ссылок на это имущество. После эмиссии имущество может передаваться между адресами с помощью трансферных транзакций (transfer transactions).

Элементы имущества, выпущенные в форме цветных монет, могут обладать многими свойствами. Они могут быть делимыми (divisible) или неделимыми (indivisible), то есть количественное выражение имущества в трансферной транзакции может быть целым числом (например, 5) или иметь вид десятичной дроби (например, 4.321). Кроме того, эмиссия элементов имущества может быть фиксированной (fixed issuance), то есть определенное количество элементов имущества выпускается только один раз, или эмиссия может быть возобновляемой (reissued), тогда после первоначального выпуска владелец прав может выпускать новые элементы имущества.

Наконец, некоторые цветные монеты позволяют получать дивиденды (dividends) в виде выплат в биткойнах, распределяемых между владельцами имущественных цветных монет пропорционально долям владения.

Транзакции цветных монет

Метаданные, придающие смысл транзакциям цветных монет, обычно хранятся в одном из фрагментов выходных данных, использующем код операции OP_RETURN. Различные протоколы цветных монет применяют разные кодировки для содержимого данных OP_RETURN. Выходные данные, содержащие OP_RETURN, называются выходными данными с маркером (marker output).

Порядок размещения выходных данных и положение выходных данных с маркером могут иметь особое значение в протоколе цветных монет. Например, в Open Access все фрагменты выходных данных, расположенные перед выходными данными с маркером, представляют выпускаемые элементы имущества. Все фрагменты выходных данных после маркера представляют передаваемые элементы имущества. Выходные данные с маркером присваивают специальные значения и цвета другим фрагментам выходных данных посредством определения их порядка в конкретной транзакции.

Другой подход применяется в Colored Coins (Colu), где выходные данные с маркером содержат код операции, который определяет, как интерпретируются метаданные. Коды операций с 0x01 по 0x0F обозначают эмиссионную транзакцию. За кодом операции эмиссии (выпуска) обычно следует идентификатор типа имущества или любой другой идентификатор, который можно использовать для извлечения информации об имуществе из какого-либо внешнего источника (например, с bittorrent). Коды операций с 0x10 по 0x1F представляют трансферную транзакцию. Метаданные трансферной транзакции содержат простые скрипты, передающие заданное количество элементов имущества из входных данных в выходные в соответствии со ссылками на их индекс. Таким образом, порядок расположения фрагментов входных и выходных данных весьма важен при интерпретации (выполнении) скрипта.

Если объем метаданных слишком велик и они не умещаются в OP_RETURN, то протокол цветных монет может использовать некоторые «хитрости» для сохранения метаданных в транзакции. Примером такой «хитрости» является размещение метаданных в погашающем скрипте (redeem script), за которым следует код операции OP_DROP, гарантирующий, что скрипт будет игнорировать эти метаданные. При другом подходе применяется скрипт мультиподписи по схеме 1-of-N, где только первый открытый ключ является действительным открытым ключом, который может расходовать выходные данные, а все последующие «ключи» заменяются закодированными метаданными.

Для правильной интерпретации метаданных в транзакции цветных монет необходимо воспользоваться совместимым кошельком или проводником блокчейна. В противном случае транзакция будет выглядеть как «обычная» биткойн-транзакция с выходными данными OP_RETURN.

Для учебного примера я создал и выпустил имущество типа MasterBTC, используя для этого цветные монеты. Имущество типа MasterBTC представляет собой ваучер на право бесплатного копирования этой книги. Ваучеры можно передавать, продавать и выкупать с помощью кошелька, совместимого с протоколом цветных монет.

Для демонстрации практического примера я воспользовался кошельком и проводником на сайте https://coinprism.info, который поддерживает протокол цветных монет Open Assets.

На рис. 12.1 показана эмиссионная транзакция в проводнике блокчейна Coinprism: https://www.coinprism.info/tx/10d7c4e022f35288779be6713471151ede967caaa39eecd35296aa36d9c109ec.

Рис. 12.1 ❖ Эмиссионная транзакция, представленная в проводнике сайта coinprism.info

На рис. 12.1 можно видеть, что проводник Coinprism показывает выпуск 20 элементов «Free copy of Mastering Bitcoin», то есть элементов имущества типа MasterBTC на специализированном адресе цветных монет:

```
akTnsDt5uzpioRST76VFRQM8q8sBFnQiwcx
```

Любые денежные средства или имущество в виде цветных монет, посланные на этот адрес, будут безвозвратно потеряны. Не отправляйте какие-либо ценности на этот адрес учебного примера.

Идентификатор этой эмиссионной транзакции представляет собой обычный идентификатор биткойн-транзакции. На рис. 12.2 показана та же транзакция в проводнике блокчейна, который не поддерживает декодирования цветных монет. Здесь используется сайт blockchain.info:

https://blockchain.info/tx/10d7c4e022f35288779be6713471151ede967caaa39eecd35296aa36d9c109ec

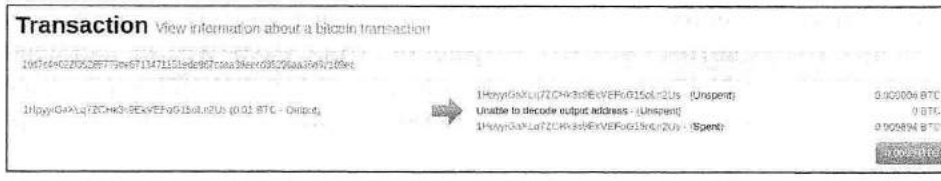


Рис. 12.2 ❖ Эмиссионная транзакция в проводнике блокчейна, который не поддерживает декодирования цветных монет

Мы видим, что blockchain.info не распознает эту транзакцию как транзакцию цветных монет. Проводник помечает второй фрагмент выходных данных предупреждением «Unable to decode output address» (Невозможно декодировать адрес для выходных данных), выделяя его красным цветом.

Если на этом экране выбрать пункт **Show scripts & coinbase** (Показать скрипты и coinbase-транзакцию), то можно увидеть более подробную информацию об исследуемой транзакции (см. рис. 12.3).

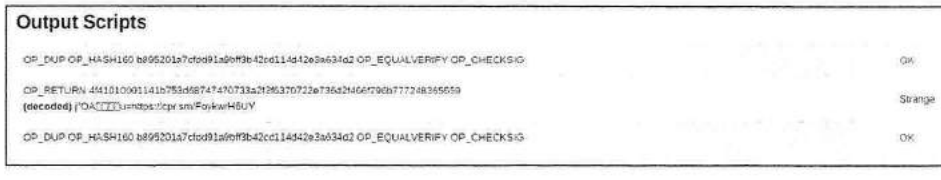


Рис. 12.3 ❖ Скрипты в эмиссионной транзакции

Но и здесь blockchain.info не распознает правильно второй фрагмент выходных данных, который помечается сообщением «Strange» (Неизвестный) и выделяется красным цветом. Тем не менее можно обнаружить, что некоторая часть метаданных в выходных данных с маркером представлена в удобочитаемой (для человека) форме:

```
OP_RETURN 4f41010001141b753d68747470733a2f2f6370722e736d2f466f796b777248365559
(decoded) "OA____u=https://cpr.sm/FoykwrH6UY
```

Попробуем извлечь содержимое транзакции с помощью утилиты командной строки `bitcoin-cli`:

```
$ bitcoin-cli decoderawtransaction `bitcoin-cli getrawtransaction
10d7c4e022f35288779be6713471151ede967caaa39eecd35296aa36d9c109ec`
```

После удаления не интересующей нас в данном случае части транзакции второй фрагмент выходных данных выглядит следующим образом:

```
{
  "value": 0.00000000,
  "n": 1,
  "scriptPubKey": "OP_RETURN
4f41010001141b753d68747470733a2f2f6370722e736d2f466f796b777248365559"
}
```

Префикс 4F41 представляет буквы OA, аббревиатуру, обозначающую Open Assets, которая помогает узнать, что следующие далее метаданные определяются протоколом Open Assets. Это строка в кодировке ASCII – ссылка на описание имущества:

```
u=https://cpr.sm/FoykwrH6UY
```

Если извлечь данные, указываемые этим URL, то получим полное описание элемента имущества в кодировке JSON, как показано ниже:

```
{
  "asset_ids": [
    "AcuRVsoa81hoLHmVTNXrRD8KpTqUXeqwgH"
],
  "contract_url": null,
  "name_short": "MasterBTC",
  "name": "Free copy of \"Mastering Bitcoin\"",
  "issuer": "Andreas M. Antonopoulos",
  "description": "This token is redeemable for a free copy of the book \"Master
ing Bitcoin\"",
  "description_mime": "text/x-markdown; charset=UTF-8",
  "type": "Other",
  "divisibility": 0,
  "link_to_website": false,
  "icon_url": null,
  "image_url": null,
  "version": "1.0"
}
```

Counterparty

Counterparty – это уровень протокола, сформированный поверх уровня биткойн-системы. Протокол Counterparty, в определенной степени похожий на протокол цветных монет, предлагает возможности создания и продажи виртуального имущества и жетонов-сертификатов. Кроме того, Counterparty предлагает функцию децентрализованного обмена имуществом. В Counterparty также реализованы смарт-контракты на основе виртуальной машины Ethereum Virtual Machine (EVM).

Как и протоколы цветных монет, Counterparty встраивает метаданные в биткойн-транзакции, используя для этого код операции OP_RETURN или адреса мультиподписи типа 1-of-N, в которых кодируются метаданные вместо открытых ключей. Применяя такие механизмы, Counterparty реализует уровень протокола, кодируемый в биткойн-транзакциях. Этот дополнительный уровень протокола может интерпретироваться совместимыми с Counterparty приложениями, такими как кошельки и проводники блокчейна, или любыми приложениями, созданными с использованием библиотек Counterparty.

Counterparty можно использовать как платформу для других приложений и сервисов. Например, Tokenly – это платформа на основе Counterparty, позволяющая авторам контента, художникам и компаниям (организациям) вы-

пускать жетоны-сертификаты (tokens), определяющие цифровые права владения, которые могут применяться для аренды, доступа, продажи или покупки контента, продукции и услуг. К другим типам приложений, использующих Counterparty, относятся игры (Spells of Genesis) и проекты распределенных грид-вычислений (Folding Coin).

Более подробную информацию о Counterparty можно найти на сайте https://counterparty.io. Проект с открытым исходным кодом размещен на сайте https://github.com/CounterpartyXCP.

Каналы платежей и каналы состояний

Каналы платежей (payment channels) представляют собой надежный механизм для обмена биткойн-транзакциями между двумя сторонами вне структуры данных блокчейна. Эти транзакции, которые должны считаться корректными, как если бы они размещались в структуре блокчейна биткойн-системы, вместо этого располагаются вне цепочки, действуя как векселя (долговые обязательства) (promissory notes) для окончательных расчетов по групповым платежам. Так как эти транзакции не оплачены (не включены в структуру блокчейна), ими можно обмениваться без обычной задержки оплаты, тем самым обеспечивая чрезвычайно высокую скорость обработки транзакций, кратковременные задержки (менее миллисекунды) и удобную степень детализации (уровня сатоши).

В действительности термин «канал» (channel) является метафорой. Каналы состояний представляют собой виртуальные конструкции для обмена состояниями между двумя сторонами вне структуры данных блокчейна. Сами по себе эти конструкции «каналами» не являются, и механизм передачи данных, заложенный в их основу, также не является каналом. Мы используем термин «канал», чтобы обозначить взаимосвязь и общее состояние между двумя сторонами вне структуры данных блокчейна.

Чтобы более глубоко понять эту концепцию, представим себе поток передачи данных по протоколу TCP. С точки зрения протоколов более высоких уровней это «сокет», соединяющий два приложения в разнородной сетевой среде (Интернет). Но если внимательно рассмотреть сетевой трафик, то можно понять, что TCP-поток – это лишь виртуальный канал на основе IP-пакетов. Каждый конечный пункт TCP-потока генерирует последовательности и выполняет сборку IP-пакетов, создавая иллюзию непрерывного потока байтов. В действительности это отдельные, никак не связанные друг с другом пакеты. Аналогично канал платежей представляет собой всего лишь последовательность транзакций. Если выстроена правильная последовательность транзакций и они соединены надлежащим образом, то создаются погашаемые долговые платежные обязательства, которым можно доверять, даже если вы не вполне доверяете другой стороне «на противоположном конце» канала.

В этом разделе мы рассмотрим различные формы каналов платежей. Сначала будут описаны механизмы, используемые для формирования односторонних (однонаправленных) каналов платежей для сервиса микроплатежей «по счет-

чику», например для потокового видео. Затем этот механизм демонстрируется в более широкой перспективе, и мы перейдем к двунаправленным каналам платежей. В конце раздела мы рассмотрим, каким образом двунаправленные каналы можно соединять последовательно для формирования многосегментных каналов в сети с маршрутизацией, получившей название Lightning Network.

Каналы платежей являются частью более широкой концепции каналов состояния (state channels), которая представляет процесс изменения состояния вне цепочки, защита которого обеспечивается в конечном итоге размещением в структуре данных блокчейна. Канал платежей – это канал состояний, в котором изменяемым состоянием является баланс виртуальных денежных средств.

Каналы состояний – основные концепции и терминология

Канал состояний устанавливается между двумя сторонами с помощью транзакции, которая блокирует совместное («разделяемое» между сторонами) состояние в структуре данных блокчейна. Такую транзакцию называют субсидирующей транзакцией (funding transaction), или анкерной транзакцией (anchor transaction). Эта единственная транзакция обязательно должна быть передана по сети и пройти процесс майнинга, чтобы создать (установить) канал. В примере с каналом платежа заблокированным состоянием является начальный баланс (денежных средств) конкретного канала.

Затем обе стороны обмениваются подписанными транзакциями, называемыми транзакциями-обязательствами (commitment transactions), которые изменяют начальное состояние. Эти транзакции являются корректными (валидными) в том плане, что они могут быть приняты к оплате любой стороной, но вместо этого сохраняются вне цепочки каждой стороной вплоть до ожидаемого закрытия канала. Обновления состояния могут возникать с той скоростью, с которой каждая сторона может создать, подписать и передать транзакцию другой стороне. На практике возможен обмен тысячами транзакций в секунду.

При обмене транзакциями-обязательствами между сторонами предыдущие состояния становятся некорректными, поэтому только самая свежая транзакция-обязательство всегда является единственной транзакцией, которую можно погасить (оплатить). Это предотвращает попытки мошенничества со стороны противоположной стороны в виде закрытия канала в одностороннем порядке с предыдущим зафиксированным состоянием, которое более выгодно этой стороне, чем текущее состояние. Мы рассмотрим различные механизмы, используемые для присваивания предыдущему состоянию статуса некорректного, ниже в этой главе.

В конечном счете канал может быть закрыт либо совместно обеими сторонами посредством записи завершающей (финальной) платежной транзакции (settlement transaction) в структуру данных блокчейна, либо в одностороннем порядке любой стороной посредством записи самой последней транзакции-обязательства в структуру данных блокчейна. Завершающая платежная транзакция представляет конечное состояние канала и фиксируется в структуре данных блокчейна.

В течение всего жизненного цикла канала только две транзакции должны быть переданы для майнинга в структуру блокчейна: субсидирующая (funding) и платежная (settlement). Между этими двумя состояниями обе стороны могут обмениваться любым количеством транзакций-обязательств, которые никогда не увидит кто-либо посторонний и которые не записываются в структуру данных блокчейна.

На рис. 12.4 изображена схема канала платежей между Бобом и Алисой, на которой показаны субсидирующая транзакция, транзакции-обязательства и платежная транзакция.

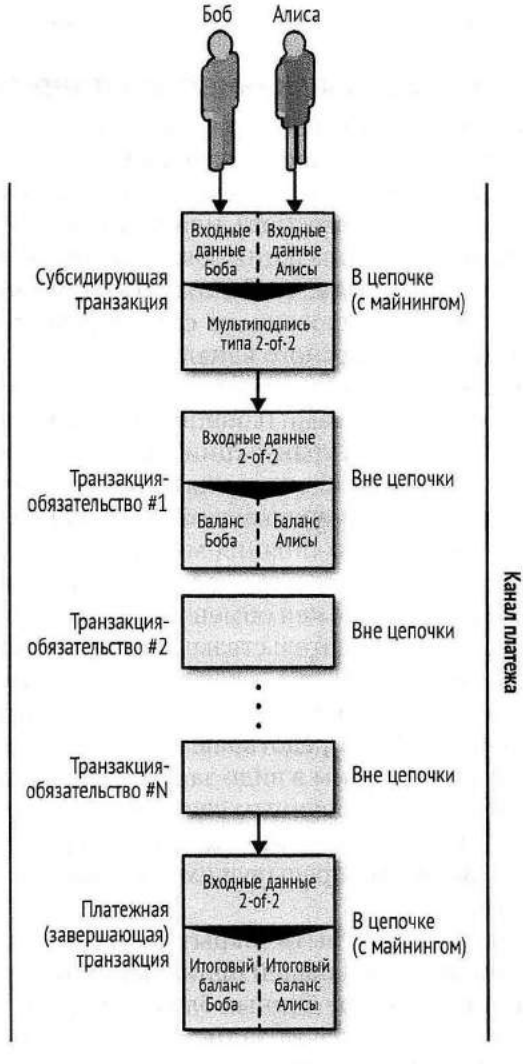


Рис. 12.4 ❖ Канал платежей между Бобом и Алисой с отображением субсидирующей транзакции, транзакций-обязательств и платежной транзакции

Пример простого канала платежей

Чтобы подробно описать каналы состояний, начнем с очень простого примера. Рассмотрим односторонний канал, в котором данные передаются только в одном направлении. Сначала для упрощения будем «наивно» предполагать, что ни одна из сторон не пытается мошенничать. После описания основного принципа работы канала рассмотрим, при каких условиях канал становится ненадежным (не вызывающим доверия), тем не менее ни одна из сторон не имеет возможности смошенничать, даже если они попытаются это сделать.

Для этого примера возьмем двух участников: Эмму (Emma) и Фабиана (Fabian). Фабиан предлагает сервис потокового видео с посекундной оплатой, для которой используется канал микроплатежей. Фабиан назначает цену 0.01 миллибита (0.00001 BTC) за секунду видео, то есть 36 миллибитов (0.036 BTC) за час видео. Эмма – пользователь, приобретающий потоковое видео на сервисе Фабиана. На рис. 12.5 изображена Эмма, оплачивающая сервис потокового видео Фабиана с использованием канала платежей.

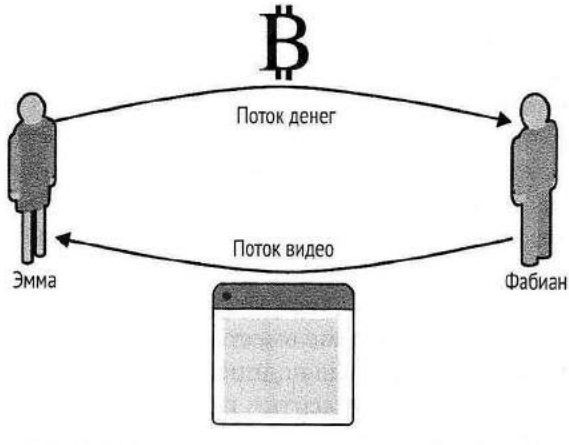


Рис. 12.5 ❖ Эмма оплачивает потоковое видео от Фабиана с помощью канала платежей, при этом платит за каждую секунду видео

В этом примере Фабиан и Эмма используют специализированное программное обеспечение, которое обеспечивает поддержку и канала платежей, и потокового видео. Эмма запускает это программное обеспечение в своем браузере, у Фабиана оно работает на сервере. Специализированное программное обеспечение обеспечивает основную функциональность биткойн-кошелька и может создавать и подписывать биткойн-транзакции. Основная концепция и собственно термин «канал платежей» полностью скрыты от пользователей. Им доступно только видео, которое оплачивается посекундно.

Для установки и настройки канала платежей Эмма и Фабиан создают адрес мультиподписи по схеме 2-of-2, и каждый хранит один из ключей. С точки зрения Эммы программное обеспечение в ее браузере представляет штриховой

QR-код с P2SH-адресом (начинающимся с цифры 3) и предлагает ей внести «депозит» за 1 час видео. Затем этот адрес субсидируется Эммой (на него переводятся соответствующие денежные средства). Транзакция Эммы с платежом на адрес мультиподписи представляет собой субсидирующую или анкерную транзакцию для канала платежа.

Продолжим рассмотрение примера и предположим, что Эмма «субсидировала» канал суммой в 36 миллибитов (0.036 BTC). Это позволит Эмме просмотреть до 1 часа потокового видео. Субсидирующая транзакция в данном случае устанавливает максимальную сумму, которая может быть передана через этот канал, то есть устанавливает пропускную способность или мощность канала (channel capacity).

Субсидирующая транзакция расходует один или несколько фрагментов входных данных из кошелька Эммы, формируя общую сумму взноса. Она создает один фрагмент выходных данных со значением 36 миллибитов, выплачиваемым на адрес мультиподписи типа 2-of-2, который совместно управляется Эммой и Фабианом. Возможно создание дополнительных фрагментов выходных данных для возврата сдачи в кошелек Эммы.

После подтверждения субсидирующей транзакции Эмма может начать просмотр потокового видео. Программное обеспечение Эммы создает и подписывает транзакцию-обязательство, которая изменяет баланс канала, кредитуя 0.01 миллибита на адрес Фабиана и возвращая 35.99 миллибита Эмме. Эта транзакция, подписанная Эммой, расходует 36 миллибитов из выходных данных, созданных субсидирующей транзакцией, и, в свою очередь, создает два фрагмента выходных данных: один – для возврата сдачи Эмме, второй – для платежа Фабиану. Эта транзакция подписана частично – требуются две подписи (схема 2-of-2), но в наличии имеется только подпись Эммы. Когда сервер Фабиана принимает эту транзакцию, он добавляет вторую подпись (для входных данных 2-of-2) и возвращает транзакцию Эмме вместе с 1 секундой оплаченного видео. Теперь у обеих сторон есть полностью подписанная транзакция-обязательство, которую каждая сторона может «погасить», представив корректный обновленный актуальный баланс канала. Стороны не публикуют эту транзакцию в сети.

В следующем раунде программное обеспечение Эммы создает и подписывает другую транзакцию-обязательство (обязательство #2), расходующую те же самые выходные данные 2-of-2 из субсидирующей транзакции. Вторая транзакция-обязательство направляет один фрагмент выходных данных с суммой 0.2 миллибита на адрес Фабиана, а второй фрагмент выходных данных с суммой 35.98 миллибита возвращает на адрес Эммы. Эта новая транзакция является оплатой двух секунд видео (в общей сложности). Программное обеспечение Фабиана подписывает и возвращает Эмме вторую транзакцию-обязательство вместе со второй секундой видео.

Точно таким же способом программное обеспечение Эммы продолжает посылать транзакции-обязательства на сервер Фабиана в обмен на секунды потокового видео. Баланс канала постепенно изменяется в пользу Фабиана, по мере того как Эмма получает очередные секунды видеопотока. Допустим,

что Эмма просматривает 600 секунд (10 минут) видео, создавая и подписывая 600 транзакций-обязательств. Самая последняя транзакция-обязательство (#600) содержит два фрагмента выходных данных, распределяющих баланс канала следующим образом: 6 миллибитов Фабиану, 30 миллибитов Эмме.

После этого Эмма щелкает по кнопке «Стоп», чтобы остановить передачу видеопотока. Теперь Фабиан или Эмма может передать конечное состояние транзакции для реальной оплаты. Эта последняя транзакция является завершающей платежной транзакцией (settlement transaction), которая выплачивает Фабиану сумму за все просмотренное Эммой видео и возвращает остаток из субсидирующий транзакции Эмме.

На рис. 12.6 показан канал между Эммой и Фабианом и транзакции-обязательства, которые обновляют баланс канала.

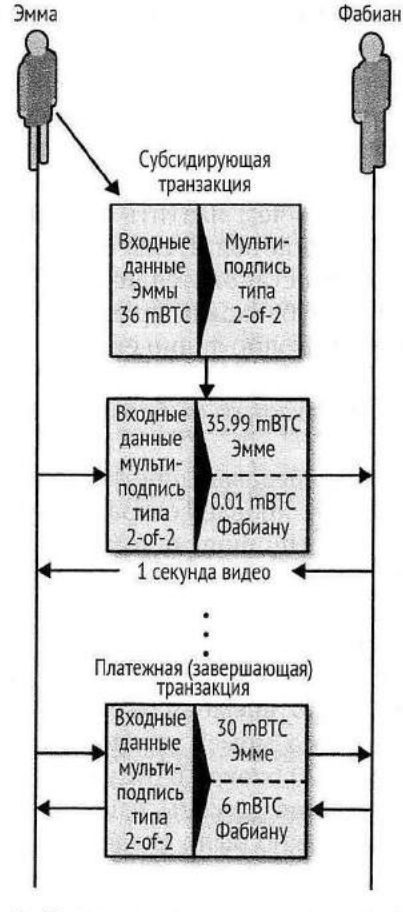


Рис. 12.6 ❖ Канал платежей между Эммой и Фабианом с отображением транзакций-обязательств, обновляющих баланс канала

В конечном итоге только две транзакции записываются в структуру данных блокчейна: субсидирующая транзакция, создающая канал платежа, и завершающая платежная транзакция, которая правильно распределяет конечный баланс между двумя участвующими сторонами.

Создание каналов без доверительных отношений

Описанный в предыдущем разделе канал работает, но только если обе стороны действуют, доверяя друг другу, без каких-либо отказов или попыток мошенничества. Рассмотрим некоторые ситуации, в которых работа канала нарушается, а также средства для устранения этих проблем:

- после выполнения субсидирующей транзакции Эмме необходимо получить подпись Фабиана, чтобы вернуть часть денег (или все деньги) обратно. Если Фабиан «исчезает», то деньги Эммы блокируются на адресе мультиподписи 2-of-2 и фактически потеряны для нее. Такой канал после создания приводит к потере денежных средств, если одна из сторон отключается от канала до того, как, по крайней мере, одна транзакция-обязательство будет подписана обеими сторонами;
- во время работы канала Эмма может взять любую транзакцию-обязательство, подписанную ею совместно с Фабианом, и передать ее в структуру данных блокчейна. Зачем платить за 600 секунд видео, если она может передать транзакцию-обязательство #1, то есть заплатить лишь за 1 секунду видео? Канал становится некорректным, потому что Эмма может смошенничать, опубликовав предыдущее (не самое последнее) обязательство, которое выгодно лично ей.

Обе эти проблемы можно устранить с помощью блокировок по времени – рассмотрим, как можно воспользоваться блокировками по времени уровня транзакции (`nLocktime`).

Эмма не может рисковать, передавая денежные средства на адрес мультиподписи 2-of-2, если ей не гарантирован возврат. Для решения этой проблемы Эмма создает субсидирующую (funding) и компенсирующую (refund) транзакции одновременно. Она подписывает субсидирующую транзакцию, но не передает ее никому. Эмма передает Фабиану только компенсирующую транзакцию и получает его подпись.

Эта компенсирующая транзакция действует как первая транзакция-обязательство, и ее блокировка по времени устанавливает верхнюю границу жизненного цикла канала. Эмма может установить для параметра `nLocktime` значение до 30 дней или до 4320 блоков в будущем. Все последующие транзакции-обязательства непременно должны иметь более короткий срок блокировки по времени, чтобы можно было погасить их ранее коменсирующей транзакции.

После того как Эмма получила полностью подписанную коменсирующую транзакцию, она может с уверенностью передать подписанную субсидирующую транзакцию, точно зная, что в конечном итоге она сможет после оконча-

ния срока блокировки погасить компенсирующую транзакцию (то есть получить обратно деньги), даже если Фабиан скроется.

Все транзакции-обязательства, которыми обмениваются стороны на протяжении жизненного цикла канала, блокируются на определенный срок в будущем. Но задержка будет немного короче для каждой следующей транзакции-обязательства, так что преждевременное погашение самого последнего обязательства становится некорректным (соответственно, и состояние канала становится некорректным). Благодаря параметру `nLocktime` ни одна из сторон не сможет успешно распространить какую-либо транзакцию-обязательство до завершения срока ее блокировки. Если все идет, как предполагалось, то обе стороны сотрудничают честно и аккуратно закрывают канал с помощью платежной транзакции (settlement transaction), устраняя необходимость (и возможность) передачи каких-либо промежуточных транзакций-обязательств. По существу, транзакции-обязательства используются только в том случае, когда одна сторона отключилась от канала, а другая сторона вынуждена корректно закрыть канал в одностороннем порядке.

Например, если для транзакции-обязательства #1 установлена блокировка на срок 4320 блоков в будущем, то транзакция-обязательство #2 блокируется на срок 4319 блоков в будущем. Для транзакции-обязательства #600 может быть израсходовано 600 блоков, прежде чем транзакция-обязательство #1 станет валидной.

На рис. 12.7 показано, как для каждой очередной транзакции-обязательства устанавливается более короткий срок блокировки по времени, позволяя расходовать ее раньше, чем предыдущие обязательства станут валидными.

Рис. 12.7 ❖ Для каждой очередной транзакции-обязательства устанавливается более короткий срок блокировки по времени, позволяя расходовать ее раньше, чем предыдущие обязательства станут валидными

Для каждой последующей транзакции-обязательства непременно должен быть установлен более короткий срок блокировки по времени, чтобы она могла распространяться в сети до ее предшественников и до компенсирующей транзакции. Возможность более раннего распространения обязательства гарантиру-

ет возможность расходования выходных данных субсидирующей транзакции и предотвращает погашение любой другой транзакции-обязательства через расходование своих выходных данных. Это обеспечивается структурой данных блокчейна биткойн-системы, исключающей двойное расходование и вводящей блокировки по времени, что позволяет каждой транзакции-обязательству без помех переводить предшественников в статус некорректных.

Каналы состояний используют блокировки по времени для выполнения смарт-контрактов с контролем времени. В этом примере мы наблюдали, каким образом контроль времени обеспечивает валидность самой последней транзакции-обязательства, отменяя все предыдущие обязательства. В результате только самая последняя транзакция-обязательство может быть передана в сеть, могут быть израсходованы только ее входные данные, а все предыдущие транзакции-обязательства переходят в статус некорректных. Выполнение смарт-контрактов с абсолютными блокировками по времени обеспечивает защиту от попыток мошенничества одной из сторон. Для реализации этой концепции не требуется ничего, кроме абсолютных блокировок по времени на уровне транзакций (`nLocktime`). Далее мы рассмотрим, как блокировки по времени на уровне скриптов – `CHECKLOCKTIMEVERIFY` и `CHECKSEQUENCEVER` – можно использовать для создания более гибких, удобных и «интеллектуальных» каналов состояний.

Реализация первой формы однонаправленного канала платежей была продемонстрирована как прототип приложения потокового видео в 2015 году группой разработчиков из Аргентины. Эту реализацию можно увидеть на сайте https://streamium.io.

Блокировки по времени не являются единственным способом перевода предыдущих транзакций-обязательств в некорректное состояние. В следующем разделе мы увидим, как можно использовать аннулирующий ключ (revocation key) для достижения той же цели. Блокировки по времени эффективны, но у них имеются два явных недостатка. При установке максимального значения блокировки по времени, когда после создания канал открывается в первый раз, такая блокировка ограничивает продолжительность жизненного цикла этого канала. И что еще хуже, блокировки по времени при практической реализации каналов вынуждают нарушать баланс между обеспечением долговременного жизненного цикла каналов и необходимостью для одного из участников очень долго ждать компенсации в случае преждевременного закрытия канала. Например, если разрешить каналу находиться в открытом состоянии 30 дней, установив этот срок для блокировки компенсации по времени, то при неожиданном исчезновении одной из сторон второй стороне придется ждать компенсации 30 дней. Чем длиннее срок блокировки, тем дольше приходится ждать компенсции.

Вторая проблема – поскольку каждая последующая транзакция-обязательство непременно должна уменьшать время блокировки, создается очевидное ограничение на количество транзакций-обязательств, которыми могут обме-

ниваться стороны. Например, канал с жизненным циклом 30 дней при установке блокировки по времени в 4320 блоков в будущем может принять только 4320 промежуточных транзакций-обязательств, после чего должен закрыться. Существует потенциальная опасность при установке интервала блокировки транзакций-обязательств длиной в 1 блок. При такой настройке разработчик создает весьма серьезную нагрузку для обеих сторон канала, которые должны постоянно «бодрствовать», оставаясь в режиме онлайн, наблюдать за текущим состоянием канала и быть готовыми к передаче корректной транзакции-обязательства в любое время.

Теперь, когда мы хорошо понимаем, как можно использовать блокировки по времени для перевода предыдущих транзакций-обязательств в статус некорректных, можно перейти к рассмотрению различий между совместным закрытием канала и закрытием в одностороннем порядке посредством опубликования в сети транзакции-обязательства. Для всех транзакций-обязательств установлены блокировки по времени, поэтому при распространении какой-либо транзакции-обязательства в сети всегда неизбежно ожидание завершения срока блокировки. Но если обе стороны достигли соглашения по окончательному балансу и точно знают, что хранят транзакции-обязательства, которые делают этот баланс действительным, то они могут создать завершающую платежную транзакцию без блокировки по времени, представляющую тот же окончательный баланс. При совместном закрытии канала каждая сторона берет самую последнюю транзакцию-обязательство и формирует практически идентичную ей платежную транзакцию, за исключением того, что блокировка по времени не устанавливается. Обе стороны могут подписать эту платежную транзакцию, зная, что нет никаких возможностей для мошенничества с целью получения более выгодного для себя баланса. После совместного подписания и передачи в сеть платежной транзакции стороны могут закрыть канал и сразу же получить суммы погашения в соответствии с окончательным балансом. В худшем случае одна из сторон может проявить мелочность, отказаться от сотрудничества и вынудить другую сторону закрыть канал в одностороннем порядке с помощью самой последней транзакции-обязательства. Но при этом самим «нарушителям» придется долго ждать получения причитающихся им денег.

Асимметричные отменяемые обязательства

Более удобным способом обработки состояний предыдущих транзакций-обязательств является их отмена в явном виде. Но этот способ не так-то просто реализовать. Главная характеристика биткойн-системы: после того как транзакция становится корректной (валидной), она остается корректной навсегда и не теряет своего статуса. Единственный способ отменить транзакцию – выполнить двойное расходование ее входных данных в другой транзакции до завершения процесса майнинга. Вот почему мы использовали блокировки по времени в простом примере канала платежей из предыдущего раздела, что-

бы обеспечить возможность расходования самых последних обязательств до того, как более старые обязательства станут корректными. Но выстраивание строгой последовательности обязательств по времени создает ряд ограничений, из-за которых практическое использование каналов платежей становится затруднительным.

Хотя транзакцию невозможно отменить, ее можно сформировать таким способом, чтобы появилась возможность признать ее нежелательной для дальнейшего использования. Для достижения этой цели необходимо предоставить каждой стороне аннулирующий ключ (revocation key) или ключ отмены, который может использоваться как средство наказания другой стороны за попытку мошенничества. Такой механизм аннулирования (отмены) предыдущей транзакции-обязательства впервые был предложен как компонент системы Lightning Network.

Для подробного рассмотрения механизма аннулирующих ключей создадим более сложный канал платежей между двумя обменными пунктами, управляемыми Хитешем (Hitesh) и Айрин (Irene). Хитеш и Айрин руководят обменными пунктами биткойнов в Индии и США соответственно. Клиенты обменного пункта Хитеша в Индии часто отправляют платежи клиентам обменного пункта Айрин в США, и наоборот. Сейчас эти транзакции фиксируются в структуре данных блокчейна биткойн-системы, но при этом приходится оплачивать транзакции и ждать завершения майнинга нескольких блоков для подтверждения. Создание канала платежей между обменными пунктами позволит сократить накладные расходы (отчисления за транзакции) и ускорить поток транзакций.

Хитеш и Айрин начинают формирование канала с совместного создания субсидирующей транзакции – от каждой стороны в канал вносится сумма 5 биткойнов. Таким образом, начальный баланс содержит 5 биткойнов для Хитеша и 5 биткойнов для Айрин. Субсидирующая транзакция блокирует начальное состояние канала с помощью мультиподписи 2-of-2 точно так же, как в предыдущем примере простого канала.

Субсидирующая транзакция может содержать один или несколько фрагментов входных данных от Хитеша (добавляющих 5 биткойнов или больше) и один или несколько фрагментов входных данных от Айрин (также добавляющих 5 биткойнов или больше). Суммы входных данных должны немного превышать мощность (емкость) канала, чтобы обеспечить оплату транзакций. Субсидирующая транзакция должна содержать один фрагмент выходных данных, блокирующий общую сумму в 10 биткойнов на адресе мультиподписи 2-of-2, управляемом Хитешем и Айрин. Субсидирующая транзакция также может содержать один или несколько фрагментов выходных данных, возвращающих сдачу Хитешу и Айрин, если суммы их входных данных больше, чем установленный взнос в канал. Это единственная транзакция с входными данными, предоставленными и подписанными обеими сторонами. Она должна быть создана в совместном режиме и подписана каждой стороной перед передачей в сеть.

Теперь вместо создания одной транзакции-обязательства, которую подписывают обе стороны, Хитеш и Айрин создают две различные транзакции-обязательства, которые являются асимметричными (asymmetric).

Хитеш создает транзакцию-обязательство с двумя фрагментами выходных данных. Первый фрагмент выходных данных выплачивает Айрин 5 биткойнов, право владения которыми она получает немедленно. Второй фрагмент выплачивает 5 биткойнов Хитешу, но право владения ими он получит только после завершения срока блокировки, равного 1000 блокам. Фрагменты выходных данных этой транзакции выглядят следующим образом:

```
Input: 2-of-2 funding output, signed by Irene

Output 0 <5 bitcoin>:
    <Irene's Public Key> CHECKSIG

Output 1:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key> CHECKSIG
```

Айрин создает другую транзакцию-обязательство с двумя фрагментами выходных данных. Первый фрагмент выходных данных выплачивает Хитешу 5 биткойнов, право владения которыми он получает немедленно. Второй фрагмент выплачивает 5 биткойнов Айрин, но право владения ими она получит только после завершения срока блокировки, равного 1000 блокам. Фрагменты выходных данных этой транзакции выглядят следующим образом:

```
Input: 2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:
    <Hitesh's Public Key> CHECKSIG

Output 1:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Irene's Public Key> CHECKSIG
```

При таком подходе у каждой стороны имеется транзакция-обязательство, расходующая выходные данные, защищенные мультиподписью 2-of-2. Эти выходные данные подписываются другой стороной. В любое время сторона, хранящая транзакцию, может также подписать ее (выполняя условие мультиподписи 2-of-2) и начать распространение в сети. Но после опубликования в сети транзакции-обязательства сторона-распространитель должна заплатить другой стороне немедленно, тогда как ей самой придется некоторое время подождать, пока не закончится срок блокировки. Из-за вынужденной задержки погашения (возврата) одного из фрагментов выходных данных для каждой стороны создаются небольшие неудобства, когда они решают в одностороннем порядке распространять в сети транзакцию-обязательство. Но одного только времени задержки недостаточно для поощрения честного поведения.

На рис. 12.8 показаны две асимметричные транзакции-обязательства, в которых платежные суммы выходных данных держателя каждого обязательства временно заблокированы (отложены на некоторое время).

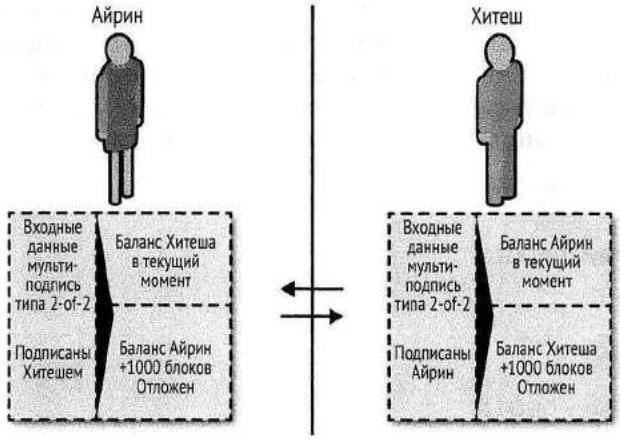


Рис. 12.8 ❖ Две асимметричные транзакции-обязательства, в которых временно заблокированы (отложены на некоторое время) платежи сторонам, хранящим эти транзакции

Рассмотрим последний элемент этой схемы: аннулирующий ключ (revocation key) или ключ отмены, который позволяет пострадавшей стороне наказать мошенника, забрав себе весь баланс канала.

Каждая транзакция-обязательство содержит «отложенные» (временно заблокированные) выходные данные. Погашающий скрипт для этих выходных данных позволяет одной стороне получить их обратно после 1000 блоков или другой стороне вернуть их при наличии аннулирующего ключа. Поэтому когда Хитеш создает транзакцию-обязательство для подписи Айрин, он делает второй фрагмент выходных данных доступным для возврата самому себе после 1000 блоков или тому, кто предъявит аннулирующий ключ. Хитеш формирует эту транзакцию и создает аннулирующий ключ, который он хранит в секрете. Он откроет аннулирующий ключ Айрин, только когда будет готов перевести канал в новое состояние и отменить эту транзакцию. Скрипт второго фрагмента выходных данных выглядит следующим образом:

```
Output 0 <5 bitcoin>:
    <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
IF
    # Revocation penalty output
    <Revocation Public Key>
ELSE
    <1000 blocks>
```

```
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key>
ENDIF
CHECKSIG
```

Айрин может без сомнений подписать эту транзакцию, поскольку если обязательство будет передано в сеть, то Айрин немедленно вернет свои деньги. Хитеш хранит транзакцию, но знает, что если он передаст ее в сеть с закрытием канала в одностороннем порядке, то ему придется ждать майнинга 1000 блоков, чтобы получить деньги.

Когда канал переходит в следующее состояние, Хитеш должен отменить (отозвать) эту транзакцию-обязательство, прежде чем Айрин согласится подписать следующую транзакцию-обязательство. Для этого Хитешу необходимо просто передать аннулирующий ключ Айрин. После того как Айрин получает аннулирующий ключ для этой транзакции, она может смело подписывать следующую транзакцию-обязательство. Теперь Айрин знает, что если Хитеш попытается смошенничать, опубликовав предыдущее обязательство, она сможет воспользоваться аннулирующим ключом, чтобы вернуть себе задержанные выходные данные Хитеша. Если Хитеш мошенничает, то Айрин получает обе суммы выходных данных.

Протокол аннулирования (отмены) является двусторонним, то есть в каждом раунде при переходе канала в новое состояние обе стороны обмениваются новыми обязательствами и подписывают транзакции-обязательства противоположной стороны. После окончательного перехода в новое состояние стороны обеспечивают невозможность использования предыдущего состояния, передавая друг другу аннулирующие ключи, необходимые для наложения взысканий за мошенничество.

Рассмотрим на практическом примере работу этого механизма. Один из клиентов Айрин хочет послать 2 биткойна одному из клиентов Хитеша. Для передачи 2 биткойнов по каналу Хитеш и Айрин должны обязательно перевести канал в следующее состояние, отображающее новый баланс. Они принимают обязательство по новому состоянию (состояние номер 2), в котором 10 биткойнов канала разделяются следующим образом: 7 биткойнов Хитешу, 3 биткойна Айрин. Чтобы достичь такого состояния канала, обе стороны создают новые транзакции-обязательства, отображающие новый баланс канала.

Как и в предыдущем примере, эти транзакции-обязательства являются асимметричными, то есть хранение транзакции-обязательства каждой стороной обеспечивает вынужденный период ожидания при попытке отмены обязательства. Важно, что перед подписью новых транзакций-обязательств стороны непременно должны сначала обменяться аннулирующими ключами, для того чтобы сделать предыдущие обязательства недееспособными (некорректными). В нашем конкретном случае Хитеш заинтересован в установлении реального состояния канала, поэтому у него нет причин для опубликования предыдущего

состояния. Но для Айрин баланс состояния номер 1 более выгоден, чем баланс состояния номер 2. Когда Айрин передает Хитешу аннулирующий ключ для ее предыдущей транзакции-обязательства (состояние номер 1), она действительно лишается возможности получить выгоду от возврата канала в более раннее состояние, так как при наличии аннулирующего ключа Хитеш может вернуть себе оба фрагмента выходных данных предыдущей транзакции-обязательства без каких-либо задержек. Таким образом, если Айрин попытается опубликовать предыдущее состояние канала, то Хитеш сможет отстоять свое право на получение всей суммы выходных данных.

Весьма важен тот факт, что отмена не выполняется автоматически. Хотя Хитеш имеет возможность наказать Айрин за мошенничество, он должен внимательно наблюдать за структурой данных блокчейна, чтобы вовремя заметить признаки обмана. Если Хитеш обнаружит публикацию предыдущей транзакции-обязательства, то у него есть интервал времени в 1000 блоков, чтобы принять меры и воспользоваться аннулирующим ключом, пресечь попытку мошенничества Айрин и наказать ее, забрав весь баланс, все 10 биткойнов.

Асимметричные отменяемые обязательства с относительными блокировками по времени (CSV) представляют собой гораздо более эффективный способ реализации каналов платежей и весьма значительное нововведение в этой технологии. При использовании этого механизма канал может оставаться открытым бесконечно и содержать миллиарды промежуточных транзакций-обязательств. В реализациях прототипов Lightning Network состояние транзакций-обязательств определялось 48-битовым индексом, обеспечивая более 281 триллиона (2.8×10^{14}) переходов между состояниями в одном канале.

Контракты Hash Time Lock Contracts (HTLC)

Функциональные возможности каналов платежей могут быть расширены с помощью специального типа смарт-контрактов, позволяющих участникам передавать денежные средства в погашающем скрипте с установкой ограничения по времени. Такой механизм называется Hash Time Lock Contract (HTLC) и используется как в двунаправленных, так и в маршрутизируемых каналах платежей.

Сначала рассмотрим значение термина «hash» в названии HTLC. Чтобы создать контракт HTLC, предполагаемый получатель платежа сначала создает секретный элемент `R`. Затем для этого секретного элемента вычисляется хэш-значение `H`:

```
H = Hash(R)
```

Полученное хэш-значение `H` может быть включено в скрипт, блокирующий выходные данные. Тот, кто знает секретный элемент, может использовать его для погашения (выплаты) выходных данных. Секретный элемент `R` также называют прообразом (preimage) для хэш-функции. Прообраз – это просто входные данные для хэш-функции.

Второй термин в названии HTLC – «time lock» (блокировка по времени). Если секретный элемент не раскрыт, то плательщик по контракту HTLC может получить «возврат» выплаченной суммы через некоторое время. Это обеспечивается с помощью механизма абсолютной блокировки по времени с использованием CHECKLOCKTIMEVERIFY.

Скрипт, реализующий контракт HTLC, может выглядеть следующим образом:

```
IF
    # Payment if you have the secret R
    HASH160 <H> EQUALVERIFY
ELSE
    # Refund after timeout.
    <locktime> CHECKLOCKTIMEVERIFY DROP
    <Payee Pubic Key> CHECKSIG
ENDIF
```

Все, кто знает секретный элемент R, который после хэширования дает значение H, могут погасить (израсходовать) эти выходные данные, выполнив первый блок управляющей конструкции IF.

Если секретный элемент не раскрыт, а контракт HTLC предъявлен, то после создания заданного количества блоков получатель может потребовать возврат денежных средств с использованием второго блока (после ELSE) управляющей конструкции IF в скрипте.

Здесь показана самая простая реализация контракта HTLC. Этот тип контракта HTLC может быть погашен любым лицом, которому известен секретный элемент R. Контракт HTLC может быть представлен во многих разнообразных формах с небольшими изменениями в скрипте. Например, добавление оператора CHECKSIG и открытого ключа в первый блок управляющей конструкции IF предоставляет право на погашение хэш-значения только конкретно указанному получателю, который также должен знать секретный элемент R.

Каналы платежа с маршрутизацией (Lightning Network)

Lightning Network – это сетевая среда с предлагаемым обеспечением маршрутизации, формируемая из двунаправленных каналов платежей, соединяемых последовательно. Подобная сеть может позволить любому участнику выполнить маршрутизацию платежа из канала в канал без какой-либо степени доверия к любому из промежуточных компонентов. Сеть Lightning Network впервые была описана Джозефом Пуном (Joseph Poon) и Тадеушем Драйа (Thadeus Dryja) в феврале 2015 года как система, основанная на концепции каналов платежей. Многие разработчики сразу поддержали эту идею и развили ее.

Lightning Network обозначает специализированное проектное решение для сети каналов платежей с маршрутизацией, которая в настоящее время реализована по меньшей мере пятью различными группами разработчиков ПО с открытым исходным кодом. Независимые реализации координируются ком-

плектом стандартов, обеспечивающих совместимость и описанных в документе Basics of Lightning Technology (BOLT) (https://github.com/lightningnetwork/lightning-rfc/blob/master/00-introduction.md).

Реализации прототипов Lightning Network были выполнены несколькими группами. В настоящее время эти реализации могут работать только в тестовой сети testnet, так как они используют технологию segwit, которая пока еще не активирована в основной структуре данных блокчейна биткойн-системы (mainnet).

Lightning Network – это лишь один из возможных способов реализации каналов платежей с маршрутизацией. Существует несколько альтернативных проектных решений, ориентированных на достижение тех же целей, например Teechan и Tumblebit.

Простой пример работы Lightning Network

Сначала рассмотрим, как работает сеть Lightning Network.

В этом примере пять участников: Алиса, Боб, Кэрол, Диана и Эрик. Здесь используются открытые каналы платежей, попарно соединяющие участников друг с другом. Алиса соединена каналом платежей с Бобом, Боб – с Кэрол, Кэрол – с Дианой, Диана – с Эриком. Для упрощения предположим, что все участники сделали взносы в размере 2 биткойнов в каждый доступный канал, следовательно, суммарная мощность каждого канала равна 4 биткойнам.

На рис. 12.9 показаны пять участников сети Lightning Network, соединенных двунаправленными каналами платежей, которые могут быть связаны в единую цепочку для передачи платежа от Алисы к Эрику (см. предыдущий раздел «Каналы платежей с маршрутизацией (Lightning Network)»).

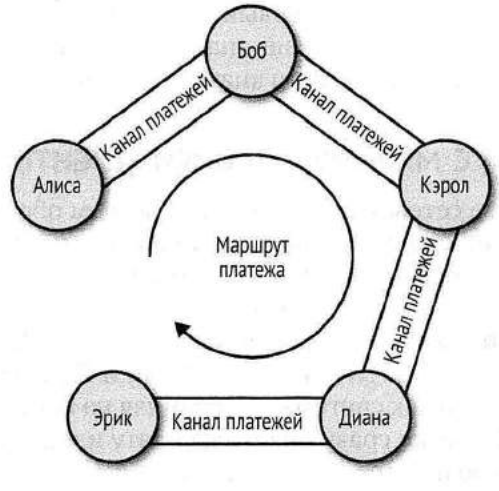


Рис. 12.9 ❖ Последовательность двунаправленных каналов платежей, связанных в сеть Lightning Network, которая может обеспечить маршрутизацию платежа от Алисы к Эрику

Алиса намерена заплатить Эрику 1 биткойн. Но у Алисы нет прямого соединения с Эриком с помощью канала платежей. Создание канала платежей требует субсидирующей транзакции, которая обязательно должна быть внесена в структуру данных блокчейна биткойн-системы. Для Алисы нежелательно открывать новый канал платежей и тратить на его субсидирование свои денежные средства. Существует ли способ непрямого платежа Эрику?

На рис. 12.10 подробно показан поэтапный процесс маршрутизации платежа от Алисы к Эрику, осуществляемый последовательностью обязательств по контрактам HTLC в каналах платежей, соединяющих участников.

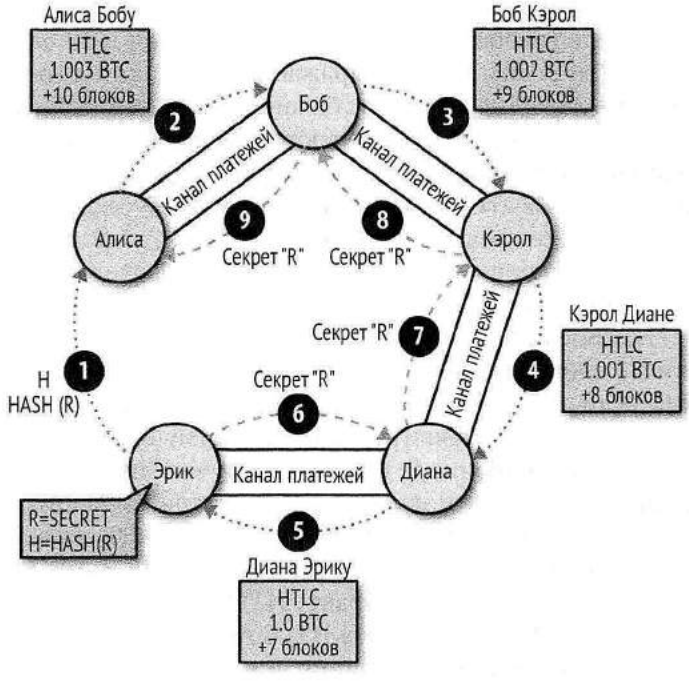


Рис. 12.10 ❖ Подробная поэтапная схема маршрутизации платежа в сети Lightning Network

Алиса инициализирует работу узла сети Lightning Network (LN), который отслеживает ее канал платежей к Бобу и обладает возможностью определения маршрутов между каналами платежей. Кроме того, узел LN Алисы может установить соединение через Интернет с узлом LN Эрика. Узел LN Эрика создает секретный элемент `R` с помощью генератора случайных чисел, но никому не раскрывает этого элемента. Узел Эрика вычисляет хэш-значение `H` для элемента `R` и передает вычисленное хэш-значение узлу Алисы (см. рис. 12.10, шаг 1).

Далее узел Алисы формирует маршрут между узлом LN Алисы и узлом LN Эрика. Более подробно алгоритм маршрутизации будет описан немного позже,

а пока просто предположим, что узел Алисы способен найти самый оптимальный маршрут.

Затем узел Алисы создает контракт HTLC, оплачиваемый по хэш-значению H с установкой тайм-аута в 10 блоков (текущий блок + 10) для возврата денег на общую сумму 1.003 биткойна (см. рис. 12.10, шаг 2). Дополнительные 0.003 биткойна будут использованы для выплаты компенсаций промежуточным узлам за их участие в обеспечении маршрутизации рассматриваемого платежа. Алиса предлагает этот контракт HTLC Бобу, вычитая 1.003 биткойна из баланса ее канала с Бобом и передавая эту сумму в контракт HTLC. Смысл контракта HTLC следующий: «Алиса снимает 1.003 биткойна со своего баланса канала для выплаты Бобу, если Боб знает секретный элемент, в противном случае эта сумма возвращается на баланс Алисы после завершения интервала ожидания в 10 блоков». Баланс канала между Алисой и Бобом теперь выражается транзакцией-обязательством с тремя фрагментами выходных данных: баланс Боба – 2 биткойна, баланс Алисы – 0.997 биткойна, 1.003 биткойна передаются в контракт HTLC Алисы. Баланс Алисы уменьшается на сумму, переданную в контракт HTLC.

Теперь у Боба есть обязательство, и если он узнает секретный элемент R за интервал создания следующих 10 блоков, то сможет предъявить права на 1.003 биткойна, заблокированные Алисой. С этим обязательством «на руках» узел Боба создает контракт HTLC в канале платежей, соединяющем его с Кэрол. Контракт HTLC Боба передает 1.002 биткойна для хэш-значения H со сроком блокировки в 9 блоков, а Кэрол сможет заявить свое право на сумму этого контракта HTLC, если знает секретный элемент R (см. рис. 12.10, шаг 3). Боб знает, что если Кэрол сможет заявить права на сумму его контракта HTLC, то ей придется определять значение R. Если Боб получит значение R за интервал создания 9 блоков, то сможет использовать его для предъявления прав на сумму контракта HTLC Алисы. Кроме того, Боб передает 0.001 биткойна для пополнения баланса канала в интервале создания девяти блоков. Если Кэрол не сможет обеспечить свои права по контракту HTLC, то Боб также не сможет заявить свое право владения по контракту HTLC Алисы, балансы обоих каналов будут возвращены в предыдущее состояние, и никто не потеряет своих денег. В рассматриваемый момент баланс канала между Бобом и Кэрол таков: 2 на балансе Кэрол, 0.998 на балансе Боба, 1.002 Боб передал в свой контракт HTLC.

После этого у Кэрол имеется обязательство, и если она узнает секретный элемент R за интервал создания следующих девяти блоков, то сможет предъявить права на 1.002 биткойна, заблокированные Бобом. Теперь Кэрол может создать обязательство по контракту HTLC в канале платежей, связывающем ее с Дианой. Контракт HTLC Кэрол передает 1.001 биткойна для хэш-значения H со сроком блокировки в восемь блоков, а Диана сможет заявить свое право на сумму этого контракта HTLC, если получит секретный элемент R (см. рис. 12.10, шаг 4). С точки зрения Кэрол, если эта операция будет выполнена правильно, она заработает 0.001 биткойна, но даже в случае неудачного завершения операции она ничего не потеряет. Ее контракт HTLC с Дианой практически осуществим

только при получении значения R, и только в этом случае она может выполнить контракт HTLC с Бобом. В рассматриваемый момент баланс канала между Кэрол и Дианой таков: 2 на балансе Дианы, 0.999 на балансе Кэрол, 1.001 Кэрол передала в свой контракт HTLC.

Наконец, Диана может предложить контракт HTLC Эрику, передавая для хэш-значения H со сроком блокировки в семь блоков (см. рис. 12.10, шаг 5). Теперь баланс канала между Дианой и Эриком распределяется следующим образом: 2 на балансе Эрика, 1 на балансе Дианы, 1 биткойн Диана передала в свой контракт HTLC.

Но на этом сегменте маршрута у Эрика есть секретный элемент R. Следовательно, он может заявить свое право владения на сумму контракта HTLC, предложенного Дианой. Эрик передает элемент R Диане и получает право на владение 1 биткойном, добавляя его к своему балансу канала (см. рис. 12.10, шаг 6). Теперь баланс канала выглядит так: 1 биткойн на балансе Дианы, 3 биткойна на балансе Эрика.

Диана получила секретный элемент R, следовательно, теперь может выполнить условия контракта HTLC с Кэрол. Диана передает элемент R Кэрол и добавляет 1.001 биткойна на свой баланс (см. рис. 12.10, шаг 7). Баланс канала между Кэрол и Дианой: 0.999 на балансе Кэрол, 3.001 на балансе Дианы. Диана «заработала» 0.001 биткойна за участие в маршрутизации рассматриваемого платежа.

Следуя по маршруту в обратном направлении, секретный элемент R позволяет каждому участнику выполнить условие отложенных контрактов HTLC. Кэрол получает 1.002 биткойна от Боба, устанавливая в канале между ними такой баланс: 0.998 на балансе Боба, 3.002 на балансе Кэрол (см. рис. 12.10, шаг 8). Наконец, Боб выполняет контракт HTLC Алисы (см. рис. 12.10, шаг 9). Баланс канала между ними обновляется следующим образом: 0.997 биткойна на балансе Алисы, 3.003 биткойна на балансе Боба.

Таким образом, Алиса заплатила Эрику 1 биткойн без открытия прямого канала платежа, связывающего ее с Эриком. При этом ни один из промежуточных пунктов в маршруте платежа не обязан доверять другим участникам. За кратковременное предоставление своих денежных средств каналам платежей участники получают возможность заработать небольшую сумму компенсации с минимальным риском, который заключается только лишь в небольшой задержке возврата денег, если канал преждевременно закрывается или возникает критический сбой в проведении маршрутизируемого платежа.

Механизмы передачи и маршрутизации в сети Lightning Network

Все каналы обмена данными между узлами LN зашифрованы в каждом сегменте, попарно соединяющем узлы. Кроме того, узлы хранят открытые ключи с длительным сроком действия, которые используются для взаимной идентификации и аутентификации между узлами.

Когда узел намеревается послать платеж другому узлу, сначала он обязательно должен сформировать маршрут (path) в сети, последовательно соединяя между собой каналы платежей с достаточной мощностью. Узлы публикуют информацию о маршрутизации, в том числе о каналах, которые они открывают, о мощности каждого канала и о сумме отчислений, выплачиваемых всем участникам маршрута. Эта информация о маршрутизации может совместно использоваться многими сторонами различным образом, при этом с большой вероятностью появляются разнообразные сетевые протоколы в форме усовершенствований технологии Lightning Network. Некоторые реализации Lightning Network применяют протокол IRC как удобный механизм публикации узлами информации о маршрутизации. Другие реализации определения маршрута пользуются пиринговой (P2P) моделью, где узлы распространяют сведения о каналах среди своих партнеров по «лавинной» (flooding) схеме, подобно распространению транзакций в биткойн-сети. В будущем планируется реализация предложения под названием Flare (http://bitfury.com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf), представляющего собой гибридную модель маршрутизации с ближайшим сетевым «соседским» окружением локального узла и удаленными на довольно значительное расстояние узлами-маяками (beacon nodes).

В нашем предыдущем примере узел Алисы использовал один из этих механизмов поиска маршрута, чтобы найти один или несколько путей, соединяющих ее узел с узлом Эрика. После того как узел Алисы сформировал маршрут, она инициализирует этот маршрут в сети, распространяя последовательность зашифрованных и вложенных инструкций для соединения каждого смежного канала платежа в единую цепочку.

Важно отметить, что этот маршрут известен только узлу Алисы. Все прочие участники маршрута платежа видят лишь смежные узлы (с которыми они соединены каналами платежей напрямую). С точки зрения Кэрол эта операция выглядит как платеж, передаваемый от Боба Диане. Кэрол ничего не знает о том, что Боб в действительности передает платеж от Алисы. Кэрол также неизвестно, что Диана перенаправит платеж Эрику.

Это очень важная функциональная характеристика сети Lightning Network, поскольку она обеспечивает секретность платежей и чрезвычайно затрудняет применение средств надзора, цензуры или внесения в черные списки. Но каким образом Алиса может сформировать этот маршрут платежа без раскрытия всей информации для промежуточных узлов?

Сеть Lightning Network реализует протокол маршрутизации Onion, обеспечивающий анонимность и основанный на схеме под названием Sphinx (http://www.cypherpunks.ca/~iang/pubs/SphinxOR.pdf; https://eprint.iacr.org/2009/628.ps). Этот протокол маршрутизации гарантирует, что отправитель платежа может сформировать и передать маршрут в сети Lightning Network следующим образом:

- промежуточные узлы могут проверить и расшифровать только доступную им часть информации о маршруте и найти следующий сегмент маршрута;

- промежуточным узлам известны только предыдущий и следующий сегменты маршрута, они не могут ничего узнать о других узлах, являющихся компонентами этого маршрута;
- промежуточные узлы не имеют возможности определить ни общую длину маршрута платежа, ни свое положение в этом маршруте;
- каждая часть маршрута зашифрована таким образом, что при атаке на сетевом уровне невозможно обнаружить взаимосвязь пакетов из различных частей маршрута платежа;
- в отличие от Tor (протокол анонимизации с Onion-маршрутизацией для Интернета), здесь нет «узлов выхода» (exit nodes), которые могут быть размещены под надзором. Нет необходимости передавать платежи в структуру данных блокчейна, узлы просто обновляют балансы каналов.

Используя этот протокол Onion-маршрутизации, Алиса «упаковывает» каждый элемент маршрута в слой шифрования, начиная с конечного элемента в обратном порядке. Она шифрует сообщение для Эрика его открытым ключом. Внешней «оберткой» для этого сообщения служит зашифрованное сообщение для Дианы с идентификацией Эрика как следующего получателя. В свою очередь, внешней «оберткой» для сообщения Диане становится зашифрованное сообщение для Кэрол с использованием открытого ключа Кэрол, а Диана идентифицируется как следующий получатель. Та же операция выполняется для сегмента Боб–Кэрол. Таким образом, Алиса сформировала своеобразную зашифрованную многослойную «луковицу» (onion) из сообщений. Она посылает эту конструкцию Бобу, который может расшифровать и «распаковать» только внешний уровень. Внутри «упаковки» Боб находит сообщение, адресованное Кэрол, которое он может переслать ей, но сам расшифровать это сообщение не может. Далее по маршруту сообщения пересылаются, расшифровываются, снова пересылаются и т. д. на протяжении всего пути к Эрику. В каждом сегменте маршрута любому участнику известны только предыдущий и следующий узлы.

Каждый элемент маршрута содержит информацию о контракте HTLC, который должен быть распространен на следующий сегмент, заданная сумма пересылается с включением соответствующего отчисления (оплаты пересылки) и с установкой времени блокировки CLTV (в блоках) до окончания срока действия контракта HTLC. По мере распространения информации о маршруте узлы передают платежи по контракту HTLC в очередной следующий сегмент.

Возможно, у читателя возникает вопрос: каким образом можно создать такое положение, при котором узлы ничего не знают о длине маршрута и о собственном положении в этом маршруте? Узлы получают сообщение и перенаправляют его в следующий сегмент. Но разве при этом сообщение не становится короче, позволяя узлам определять длину маршрута и свою позицию в нем? Чтобы сохранить секрет, длина маршрута всегда имеет фиксированную длину в 20 сегментов и при необходимости дополняется случайными данными. Каждый узел видит следующий сегмент и зашифрованное сообщение постоянной длины, которое необходимо передать дальше. Только конечный получатель

обнаруживает отсутствие следующего сегмента. Для всех прочих узлов видимая длина маршрута всегда равна 20 сегментам.

Преимущества сети Lightning Network

Lightning Network представляет собой технологию маршрутизации второго уровня (по отношению к биткойн-сети). Ее можно применять к любой структуре данных блокчейна, которая поддерживает некоторые основные функциональные возможности, такие как транзакции с мультиподписями, блокировки по времени и простые смарт-контракты.

Если Lightning Network размещается поверх биткойн-сети, то биткойн-сеть получает возможность существенного улучшения мощности, приватности, детализации и скорости без отступления от принципов выполнения действий на основе доверительных отношений без участия посредников:

- приватность – платежи в сети Lightning Network с точки зрения приватности защищены гораздо лучше, чем платежи в структуре данных блокчейна, так как не являются открытыми для всех. Участники маршрута могут видеть платежи, проводимые через их каналы, но они ничего не знают об отправителе и получателе;
- взаимозаменяемость – Lightning Network существенно усложняет установление надзора (наблюдения) и применение черных списков (blacklists) в биткойн-системе, тем самым улучшая взаимозаменяемость цифровых валют;
- скорость – биткойн-транзакции с использованием Lightning Network исполняются за миллисекунды (а не за минуты), так как клиринг контрактов HTLC осуществляется без передачи транзакций в блок;
- высокая детализация – Lightning Network обеспечивает платежи с такими малыми суммами, как минимально допустимая биткойн-«пылинка», возможно, даже с более мелкими суммами. Кое-кто предлагает ввести расчеты в субсатоши (subsatoshi);
- мощность – Lightning Network увеличивает мощность биткойн-системы на несколько порядков. Практически не существует верхнего предела для количества платежей в секунду с маршрутизацией через Lightning Network, поскольку это количество зависит только от мощности и скорости каждого узла;
- операции без взаимного доверия сторон – Lightning Network использует биткойн-транзакции между узлами, функционирующими как равноправные партнеры без доверия друг другу. Таким образом, Lightning Network сохраняет принципы биткойн-системы, но существенно улучшает функциональные параметры.

Как уже было сказано раньше, Lightning Network, конечно же, не является единственным способом реализации каналов платежей с маршрутизацией. Предлагаются и другие системы, например Tumblebit и Teechan. Но в настоящее время система Lightning Network уже развернута в тестовой сети testnet.

Несколько различных групп уже разработало альтернативные реализации Lightning Network и работает над обеспечением требований стандарта общей взаимозаменяемости (стандарт BOLT). Вероятнее всего, именно сеть Lightning Network будет первой сетью каналов платежей с маршрутизацией, развернутой для реальной эксплуатации.

Резюме

Мы рассмотрели лишь несколько готовых к эксплуатации приложений, которые могут использовать структуру данных блокчейна биткойн-системы как платформу для создания доверительных отношений. Эти приложения расширяют область практического применения биткойна, выводя ее за рамки платежных систем и финансовых инструментов и привлекая в эту область множество других типов приложений, для которых очень важны доверительные отношения. Осуществляя децентрализацию основы доверительных отношений, структура данных блокчейна биткойн-системы представляет собой платформу, способную породить множество действительно революционных приложений в широком спектре отраслей промышленности.

Приложение A

Статья о биткойне Сатоши Накамото

Это оригинал статьи, воспроизводимый здесь в том виде, в котором статья была опубликована Сатоши Накамото в октябре 2008 года.

Биткойн – пиринговая система электронных денег

Сатоши Накамото (*Satoshi Nakamoto*)
satoshin@gmx.com
www.bitcoin.org

Аннотация. Чисто пиринговая версия электронных денег предназначена для обеспечения выполнения онлайновых платежей, отправляемых напрямую от одной стороны другой стороне без проведения через финансовые учреждения. Цифровые подписи частично решают проблему, но главные преимущества теряются, если по-прежнему требуется заслуживающая доверия третья сторона, чтобы предотвратить двойное расходование. Мы предлагаем решение проблемы двойного расходования с помощью пиринговой сети. Сеть обеспечивает включение меток времени в транзакции и размещение хэш-значений транзакций в постоянно наращиваемой цепочке доказательств выполнения работы на основе хэш-функций, формируя запись, котору невозможно изменить без повторного доказательства выполнения работы. Самая длинная цепочка не только служит подтверждением засвидетельствованной последовательности событий, но также является доказательством того, что источником ее является самый крупный пул мощностей CPU. Пока подавляющее большинство мощностей CPU управляется узлами, которые не объединяются для атаки на сеть, узлы будут генерировать самую длинную цепочку и всегда опережать атакующих. Сама сеть требует минимальной структуризации. Сообщения распространяются силами самих участников сети, а узлы могут отключаться от сети и вновь присоединяться к ней в любое время, принимая самую длинную цепочку доказательств выполнения работы как подтверждение всех событий, происшедших за время их отсутствия.

Введение

Коммерческая деятельность в Интернете вынуждена почти полностью полагаться исключительно на финансовые организации, действующие как заслуживающие доверие третьи стороны в процессе электронных платежей. Несмотря на то что эта система достаточно хорошо работает для большинства транзакций, ей все же присущи недостатки, наследуемые от модели, основанной на полном доверии. Абсолютно необратимые транзакции в действительности невозможны, так как финансовые организации не могут избежать разногласий при посредничестве. Оплата посредничества увеличивает стоимость транзакций, ограничивая минимальный размер транзакции на практике и лишая клиентов возможности проведения нерегулярных отдельных транзакций с малыми суммами. Также имеет место рост стоимости при потере возможности выполнения невозвращаемых платежей за необратимые услуги. Для обеспечения возможности возврата необходимо распространение доверительных отношений. Продавцы вынуждены с подозрением относиться к своим клиентам, запрашивая у них гораздо больший объем информации, чем требуемый объем информации при полном доверии. Определенный процент случаев мошенничества принимается как неизбежное явление. Подобной ненадежности стимостей и платежей можно было бы избежать при личном использовании физических денежных знаков, но не существует механизма выполнения платежей через каналы обмена данными без участия заслуживающих доверия третьих сторон.

Поэтому необходима электронная платежная система, основанная на криптографическом доказательстве, а не на доверительных отношениях, позволяющая двум добровольно действующим сторонам выполнить транзакцию напрямую между этими сторонами без необходимости привлечения третьей стороны, заслуживающей доверия. Транзакции, которые с точки зрения вычислимости практически невозможно реверсировать или отменить, должны защитить продавцов от мошенничества, а подпрограмма реализации механизмов депонирования может быть без затруднений реализована для защиты покупателей. В этой статье мы предлагаем решение проблемы двойного расходования с использованием пиринговой распределенной системы с сервером меток времени для генерации вычисляемого доказательства хронологического порядка транзакций. Система безопасна, пока честно действующие узлы совместно управляют большей частью вычислительной мощности CPU, по сравнению с любой объединившейся группой атакующих узлов.

Транзакции

Мы определяем электронную монету (денежный знак) как цепочку цифровых подписей. Каждый владелец передает денежный знак следующему владельцу с помощью цифровой подписи хэш-значения предыдущей транзакции и открытого ключа следующего владельца посредством добавления этих данных к концу передаваемого денежного знака. Получатель платежа может проверить подписи для проверки всей цепочки прав владения.

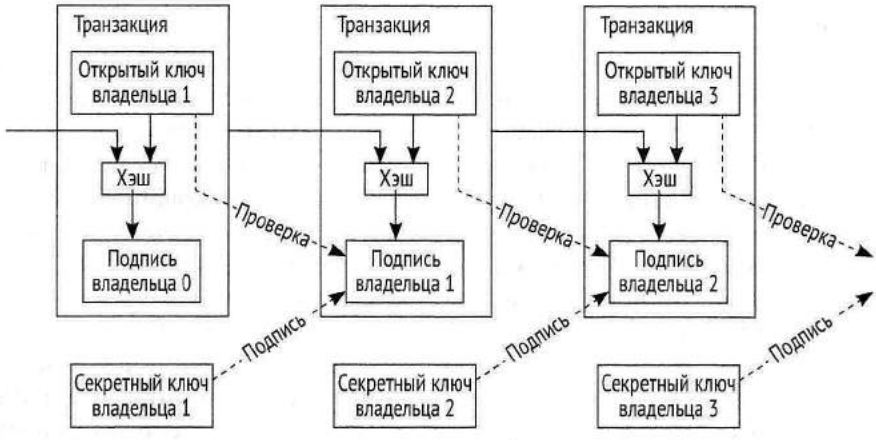


Рис. А.1

Разумеется, проблема заключается в том, что получатель платежа не может проверить тот факт, что один из владельцев не совершил двойное расходование этой передаваемой денежной единицы. Общее решение – ввод заслуживающего доверия центрального органа авторизации или «монетного двора» (министерства финансов), который проверяет каждую транзакцию на двойное расходование. После каждой транзакции денежный знак должен быть возвращен на монетный двор для выпуска нового денежного знака, и только денежные знаки, вышедшие непосредственно с монетного двора, заслуживают полного доверия, и можно быть уверенным в том, что они не расходуются дважды. Проблема такого решения заключается в том, что судьба всей денежной системы зависит от компании, владеющей монетным двором, а также в том, что каждая транзакция должна проходить через эту компанию, действующую как банк.

Необходим способ, при котором получатель платежа точно знает, что предыдущие владельцы не подписывали каких-либо более ранних транзакций. Для наших целей более ранние транзакции (по отношению к текущей) всегда учитываются, поэтому мы можем не беспокоиться о последующих попытках двойного расходования. Единственным способом подтверждения отсутствия какой-либо транзакции является отслеживание всех транзакций. В модели «монетного двора» центральному финансовому органу известны все транзакции, и он решает, какая транзакция была выполнена первой. Для решения этой задачи без третьей доверенной стороны транзакции должны быть опубликованы и открыты для всех [1], и нам необходима система, в которой участники достигают соглашения о единой хронологии порядка поступления (фиксации) транзакций. Получателям платежей необходимо подтверждение времени выполнения каждой транзакции, а большинство узлов подтверждает, что конкретная транзакция была принята первой.

Сервер меток времени

Описание предлагаемого нами решения следует начать с сервера меток времени. Сервер меток времени принимает хэш-значение блока элементов, которому должна быть присвоена метка времени, после чего открыто публикует помеченное хэш-значение, подобно новостному материалу или посту в Usenet [2–5]. Метка времени гарантирует, что данные действительно существовали в указанное время, очевидно, в порядке их включения в хэш-значение. Каждая метка времени содержит предыдущую метку времени в своем хэше, образуя цепочку, в которой каждая добавляемая метка времени закрепляет правильность предыдущей метки.

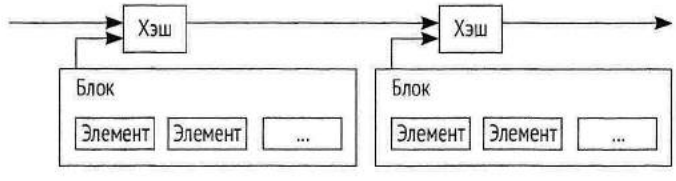


Рис. А.2

Доказательство выполнения работы

Для реализации распределенного сервера меток времени в пиринговой системе потребуется использование подсистемы доказательства выполнения работы (proof-of-work), подобной системе Hashcash Адама Бэка (Adam Back) [6], отличающейся от принципов организации службы новостей или постов в конференции Usenet. Доказательство выполнения работы подразумевает поиск значения, которое после хэширования, например с помощью алгоритма SHA-256, дает хэш-значение, начинающееся с определенного количества нулевых битов. Средний объем работы, требуемый для решения этой задачи, находится в экспоненциальной зависимости от заявленного количества начальных нулевых битов, а правильность решения можно проверить выполнением одной операции хэширования. Для нашей сети с метками времени реализован алгоритм доказательства выполнения работы с инкрементированием дополнительного случайного значения nonce в блоке до тех пор, пока не будет найдено значение, которое дает хэш-значение блока с требуемым количеством начальных нулевых битов. После того как вычислительные ресурсы CPU действительно были затрачены на получение результата, соответствующего условию доказательства выполнения работы, блок не может быть изменен без повторного выполнения того же объема работы. После добавления в цепочку новых блоков при попытке изменения рассматриваемого блока общий объем работы должен включать повторное выполнение доказательства для всех последующих блоков.

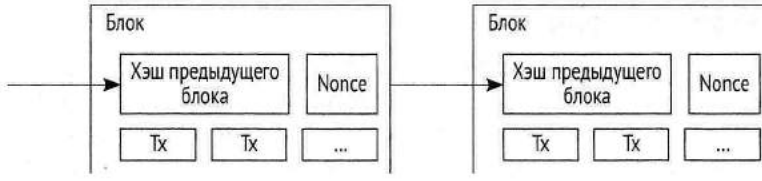


Рис. А.3

Доказательство выполнения работы также решает проблему определения большинства при принятии решений. Если бы определение большинства было основано на принципе «один IP-адрес – один голос», то лица, имеющие возможность создания множества IP-адресов, могли бы нарушить корректность процедуры голосования. В сущности, подсистема доказательства выполнения работы реализует принцип «один CPU – один голос». Решение большинства представлено самой длинной цепочкой, на которую затрачен самый большой объем вычислительных затрат по доказательству выполнения работы. Если большинство вычислительных мощностей CPU управляется узлами, действующими честно, то самая корректная («честная») цепочка будет расти наиболее быстро и обгонять все конкурирующие цепочки. Для изменения какого-либо ранее созданного блока атакующий должен будет проделать заново весь объем по доказательству выполнения работы для этого блока и для всех блоков после него, то есть догнать и обогнать честно работающие узлы. Ниже мы покажем, что для атакующего с меньшими вычислительными ресурсами вероятность достижения уровня всех узлов сети экспоненциально уменьшается по мере добавления последующих блоков.

Для компенсации постоянно увеличивающейся скорости работы аппаратуры и регулирования заинтересованности в постоянной работе узлов сложность задачи доказательства выполнения работы определяется измененяющимся средним целевым значением в виде среднего количества блоков в час. Если блоки генерируются слишком быстро, уровень сложности увеличивается.

Сеть

Фазы работы сети описаны ниже:

1. Новые транзакции распространяются по всем узлам.
2. Каждый узел собирает новые транзакции в блок.
3. Каждый узел начинает поиск решения сложной задачи доказательства выполнения работы для этого блока.
4. Когда узел находит решение задачи доказательства выполнения работы, он распространяет свой блок по всем узлам.
5. Узлы принимают предлагаемый блок, только если все транзакции в нем корректны (валидны) и пока еще не израсходованы.
6. Узлы выражают свое согласие с приемом предложенного блока, начиная процесс создания следующего блока в цепочке и используя хэш-значение принятого блока как предыдущее хэш-значение.

Узлы всегда рассматривают самую длинную цепочку как корректную и продолжают работу по ее наращиванию. Если два узла одновременно распространяют различные версии следующего блока, то некоторые узлы могут принять первой одну из этих версий. В этой ситуации узлы продолжают обработку блока, принятого первым, но сохраняют и другую ветвь на тот случай, если она станет более длинной. Конфликт может быть разрешен, если найдено очередное доказательство выполнения работы и одна из ветвей становится более длинной. После этого узлы, которые работали над другой ветвью, переключаются на более длинную ветвь.

Для новых распространяемых транзакций достижение абсолютно всех узлов сети не является обязательным условием. После поступления на большинство узлов транзакции сразу же включаются в очередной создаваемый блок. Схема распространения блоков также защищена от потери сообщений. Если узел не получил блок, то он затребует его после получения следующего блока, когда обнаружит, что один блок пропущен.

Стимул

По соглашению самая первая транзакция в любом блоке является особой транзакцией, которая создает новую денежную единицу (монету), принадлежащую создателю блока. Это дополнительный фактор, стимулирующий поддержку узлами работы сети и создающий возможность изначального распределения денежных средств, вводимых в обращение при отсутствии какого-либо центрального органа управления, выпускающего дензнаки. Стабильное добавление постоянного количества новых денежных единиц представляет собой аналогию с процессом добычи золота, при котором расходуются ресурсы для добавления некоторого объема золота в обращение. В нашем случае ресурсы – это процессорное (CPU) время и затраченная электроэнергия.

Кроме того, можно создать стимул в форме оплаты транзакций. Если сумма выходных данных транзакции меньше суммы входных значений, то разность становится оплатой транзакции, которая добавляется к поощрительной сумме блока, содержащего эту транзакцию. Так как в обращение вводится ограниченное предопределенное количество денежных единиц, со временем стимулирующий фактор может полностью перейти к отчислениям за транзакции и полностью устранить инфляцию.

Стимулирующий фактор может помочь узлам оставаться честными, поощряя их за это. Если корыстолюбивый мошенник сможет сконцентрировать больше вычислительных ресурсов CPU, чем все честные узлы, то ему придется сделать выбор: использовать свою мощь для обмана людей и похищения их платежей или воспользоваться ею для генерации новых денег. Ему придется признать, что более выгодно играть по правилам, поскольку по этим правилам он получает преимущество при создании новых денежных единиц, по сравнению с конкурентами, а не нарушать работу системы и наносить ущерб собственному благосостоянию.

Требуемое дисковое пространство

После того как самая последняя транзакция в денежной системе защищена сверху достаточным количеством блоков, необходимо проведение расходующих транзакций, прежде чем можно будет удалить эту транзакцию для экономии дискового пространства. Чтобы сделать это без нарушения структуры хэш-значения блока, хэш-значения транзакций сохраняются в дереве Меркле (Merkle Tree) [7] [2] [5], а в хэш-значение блока включается только корень дерева. Таким образом, старые блоки со временем могут быть сжаты за счет укорочения ветвей дерева. При этом нет необходимости хранить внутренние хэш-значения.

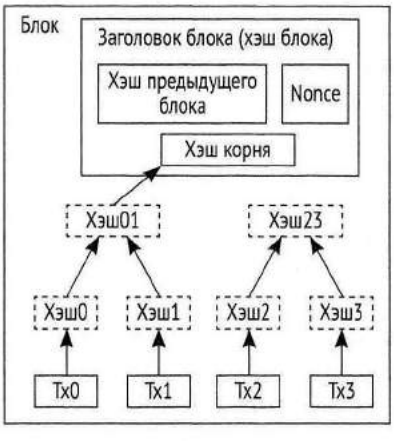


Рис. А.4 ❖ Хэш-значения транзакций, сохраненные в дереве Меркле

Рис. А.5 ❖ Состояние того же блока после удаления из него ветви Tx0-2

Размер заголовка блока без транзакций приблизительно равен 80 байтам. Предположим, что блоки генерируются через каждые 10 минут, тогда потребуется дисковое пространство `80 байтов * 6 * 24 * 365 = 4.2 Мб` в год. В 2008 году обычные настольные компьютеры общего назначения комплектуются 2 Гб оперативной памяти, а по закону Мура можно предсказать стабильный рост по 1.2 Гб в год, поэтому хранение данных не должно стать проблемой, даже если заголовки блоков нужно будет обязательно хранить в оперативной памяти.

Упрощенная верификация платежей

Существует возможность проверки (верификации) платежей без ввода в эксплуатацию полноценного сетевого узла. Пользователь должен хранить только копию заголовков блоков самой длинной цепочки доказательства выполнения работы, которую можно получить по запросу к сетевым узлам, которые уже вполне уверены в том, что хранят самую длинную цепочку и получили ветвь дерева Меркле, связывающую проверяемую транзакцию с блоком с соответствующей ей меткой времени. Пользователь не может проверить саму транзакцию, но, прослеживая связь этой транзакции с ее местоположением в цепочке, он может убедиться в том, что сетевой узел принял ее, а блоки, добавленные после соответствующего блока, подтверждают, что эта транзакция принята всей сетью.

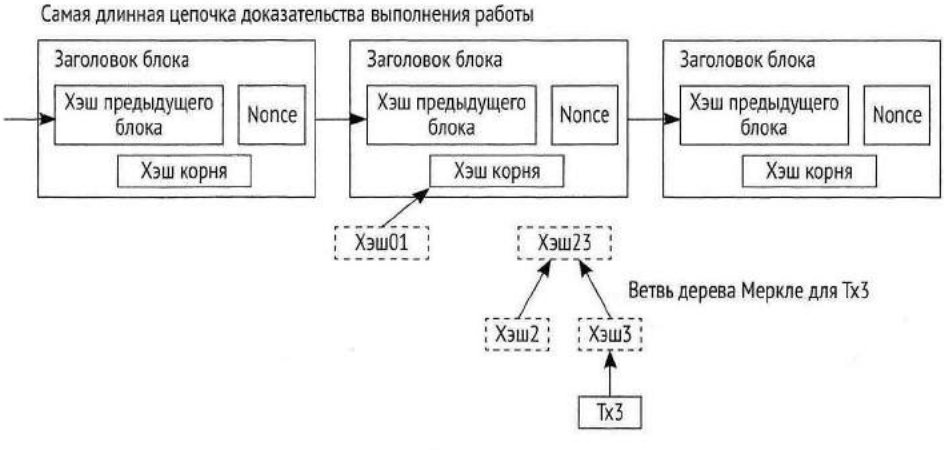


Рис. А.6

По этой схеме верификация является надежной, пока честные узлы управляют сетью, но она более уязвима, если сеть захвачена атакующим мошенником. В то время как полноценные сетевые узлы могут проверять сами транзакции, проверку по упрощенной методике можно проводить обманным путем с помощью транзакций, сфабрикованных атакующим, в течение всего времени, пока мошенник продолжает контролировать сеть. Одной из стратегий защиты

против такой атаки может стать прием предупреждений от узлов сети, когда они обнаруживают некорректный (невалидный) блок и рекомендуют пользовательскому программному обеспечению загрузить этот блок полностью со всеми подозрительными транзакциями, чтобы подтвердить их несогласованность. Владельцы бизнес-предприятий, которые часто получают платежи, вероятнее всего, острее почувствуют необходимость создания собственного полноценного сетевого узла для обеспечения более независимой от других системы защиты и поддержки более быстрой верификации.

Объединение и разделение сумм транзакций

Несмотря на потенциальную возможность отдельной обработки денежных единиц, создание отдельной транзакции для каждого передаваемого цента, скорее всего, будет чрезвычайно неуклюжим решением. Чтобы обеспечить разделение и объединение сумм, транзакции содержат несколько фрагментов входных и выходных данных. Обычно будут возникать варианты с одним фрагментом входных данных, взятых из более крупной предыдущей транзакции, или варианты с несколькими фрагментами входных данных, объединяющих более мелкие суммы, и в большинстве случаев два фрагмента выходных данных: один – для платежа, второй – для возврата сдачи отправителю, если это необходимо.

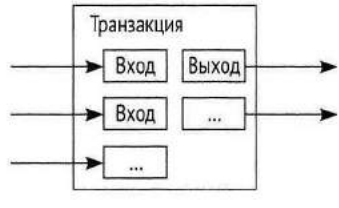


Рис. А.7

Здесь следует отметить, что такое разветвление, когда транзакции зависят от нескольких других транзакций, которые, в свою очередь, также зависят от многих других транзакций, не создает никаких проблем. Необходимость в извлечении полной отдельной копии хронологии транзакций не возникает никогда.

Приватность

Привычная для большинства людей модель банковского обслуживания обеспечивает необходимый уровень приватности, разрешая доступ к информации только сторонам-участникам и заслуживающей доверия третьей стороне. Необходимость открытой публикации всех транзакций исключает возможность применения этого метода, но приватность может быть сохранена при помощи разрыва потока информации в другом месте: сохранение анонимности открытых ключей. Все могут видеть, что некто отправляет какую-то сумму кому-то

другому, но нет никакой информации о связи этой транзакции с кем-либо. Это похоже на уровень доступности информации, публикуемой фондовыми биржами, где открыто объявляются время и размер отдельных сделок, «лента» (tape), но ничего не сообщается об участниках этих сделок.

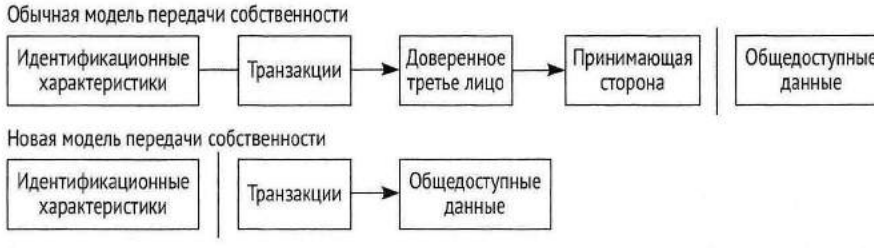


Рис. А.8

В качестве дополнительной защиты следует использовать новую пару ключей для каждой транзакции, чтобы исключить возможность установления связи конкретной транзакции с одним из известных владельцев. Некоторых возможностей установления такой связи избежать не удается в транзакциях с несколькими фрагментами входных данных, в которых необходимо раскрыть информацию о том, что определенные фрагменты принадлежали одному и тому же владельцу. Здесь опасность заключается в том, что если владелец ключа становится известным, то появляется возможность обнаружить другие транзакции, принадлежащие этому же владельцу.

Вычисления

Мы рассматриваем сценарий, при котором атакующий пытается сгенерировать альтернативную цепочку быстрее, чем формируется основная цепочка честных участников. Даже если такая попытка удается, это не делает систему открытой для любых произвольных изменений, таких как создание ценностей «из воздуха» и присваивание денег, которые никогда не принадлежали атакующему мошеннику. Узлы не принимают некорректную транзакцию в качестве платежа, а честно действующие узлы никогда не примут блок, содержащий некорректные транзакции. Атакующий может лишь попытаться изменить одну из своих транзакций, чтобы вернуть недавно потраченные деньги.

Конкуренцию между честно созданной цепочкой и цепочкой атакующего можно охарактеризовать как биномиальное случайное блуждание (Binomial Random Walk). Успешное событие – расширение честно созданной цепочки на один блок, увеличивающее ее лидерство на +1, а отрицательное событие – расширение цепочки атакующего на один блок, сокращающее разрыв между цепочками на –1.

Определение вероятности того, что атакующий догонит «честную» цепочку при заданном отставании, аналогично задаче о разорении игрока (Gambler's

Ruin). Предположим, что игрок с неограниченным кредитом начинает игру с дефицитом (отставанием) и играет потенциально бесконечное количество раундов, пытаясь добиться выигрышного для себя результата. Мы можем вычислить вероятность достижения такого результата, то есть вероятность того, что атакующий догонит «честную» цепочку, следующим образом [8]:

- p == вероятность того, что «честная» цепочка создает очередной блок;
- q == вероятность того, что атакующий создает следующий блок;
- q_z == вероятность того, что атакующий все-таки догонит «честную» цепочку при начальном отставании в z блоков:

$$q_z = \begin{cases} 1 & \text{если } p \le q \\ (q/p)^z & \text{если } p > q \end{cases}.$$

Предположим, что p > q, тогда вероятность экспоненциально уменьшается при увеличении количества блоков, которые должен создать атакующий, чтобы догнать «честную» цепочку. Если атакующий не сделает успешный рывок вперед как можно раньше при таких неблагоприятных условиях, то его шансы становятся исчезающе малыми, по мере того как отставание увеличивается.

Теперь рассмотрим, какое время ожидания необходимо для получателя новой транзакции, чтобы быть достаточно уверенным в том, что отправитель не сможет изменить эту транзакцию. Предположим, что отправитель – это атакующий, который хочет заставить получателя на некоторое время поверить в то, что отправитель действительно платит ему, а затем вернуть себе платеж через некоторое время. Получатель получит оповещение об этом возврате, но получатель надеется, что оповещение придет слишком поздно.

Получатель генерирует новую пару ключей и передает отправителю открытый ключ непосредственно перед подписью. Это позволяет помешать отправителю заранее подготовить цепочку блоков, обрабатывая ее постоянно до тех пор, пока определенная доля удачи не позволит ему вырваться вперед в гонке цепочек, а затем выполнить свою транзакцию в этот наиболее выгодный момент. После отправки транзакции нечестный отправитель начинает тайную работу над параллельной цепочкой, содержащей альтернативную версию своей транзакции.

Получатель ждет до тех пор, пока транзакция не будет добавлена в блок, после чего z блоков должны быть присоединены к цепочке после этого блока. Получателю неизвестно, насколько успешно продвигается работа атакующего, но если предположить, что для создания «честных» блоков требуется среднее расчетное время на блок, то потенциальный прогресс в работе атакующего будет подчиняться закону распределения Пуассона с ожидаемым значением:

$$\lambda = z\frac{q}{p}.$$

Чтобы вычислить вероятность, с которой атакующий все еще может догнать «честную» цепочку в текущий момент, умножим распределение Пуассона для

каждой величины количественной оценки прогресса, которой атакующий может достичь, на вероятность, с которой он мог бы догнать «честную» цепочку, начиная с этого момента:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{если } k \le z \\ 1 & \text{если } k > z \end{cases}.$$

После преобразования этой формулы, чтобы избежать суммирования бесконечной последовательности распределения, получаем:

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} (1 - (q/p)^{(z-k)}).$$

Реализация полученной формулы 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;
}
```

Выполняя эту программу для получения нескольких вариантов результатов для различных исходных данных, можно видеть, что вероятность экспоненциально уменьшается при увеличении значения 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
```

Решение для P, меньшего чем 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
```

Резюме

Мы предложили систему для выполнения электронных транзакций без зависимости от какой-либо степени доверия. Мы начали описание с обычной программной среды поддержки денежной системы на основе цифровых подписей, которая обеспечивает жесткое управление правами владения, но такая система является незавершенной без реализации методики исключения двойного расходования. Для решения этой проблемы мы предложили пиринговую сеть с использованием алгоритма доказательства выполнения работы для сохранения общедоступной хронологии транзакций, которая с точки зрения объема вычислений быстро становится практически неуязвимой для атак с целью изменения, если честно действующие узлы управляют большей частью вычислительных мощностей CPU. Предложенная сеть надежна и устойчива при своей неструктурированной простоте. Узлы с самого начала работают с минимальным уровнем координации. Им не нужна идентификация, поскольку сообщения не маршрутизируются в какой-либо конкретный пункт, необходима только доставка сообщений на основе максимальной эффективности (по принципу «лучшее из возможного»). Узлы могут отключаться от сети и снова присоединяться к ней в любой момент, принимая самую длинную цепочку доказательств выполнения работы как подтверждение всех событий, происшедших за время их отсутствия. Узлы голосуют с помощью своей вычислительной мощности CPU, подтверждая прием корректных (валидных) блоков работой по наращиванию цепочки, и отвергают некорректные блоки, прекращая их обработку. Все необходимые правила и стимулы могут быть введены с помощью этого механизма консенсуса.

Ссылки

[1] *W. Dai.* b-money. URL: 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. № 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. URL: 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

Эта статья опубликована в октябре 2008 года автором Сатоши Накамото. В дальнейшем (в 2009 году) она была добавлена как сопроводительная документация к программному обеспечению биткойна и защищена лицензией MIT. Статья воспроизводится в данной книге без каких-либо изменений, за исключением некоторых особенностей форматирования (и перевода), и также защищена лицензией MIT:

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, ИМЕЮЩИМ ПРИЧИНОЙ ИЛИ СВЯЗАННЫМ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ ИЛИ ИСПОЛЬЗОВАНИЕМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИЛИ ИНЫМИ ДЕЙСТВИЯМИ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ.

Приложение Б

Операторы, константы и символы скриптового языка для транзакций Script

Таблицы и описания, взятые с сайта https://en.bitcoin.it/wiki/Script.

В табл. Б.1 описаны операторы, предназначенные для записи значений в стек.

Таблица Б.1. Запись значений в стек

Символ	Значение (16-ричное)	Описание
OP_0 или OP_FALSE	0x00	Пустой массив помещается в стек
1–75	0x01–0x4b	Поместить следующие N байтов в стек, где N – от 1 до 75 байтов
OP_PUSHDATA1	0x4c	Следующий байт скрипта содержит число N, поместить следующие N байтов в стек
OP_PUSHDATA2	0x4d	Следующие два байта скрипта содержат число N, поместить следующие N байтов в стек
OP_PUSHDATA4	0x4e	Следующие четыре байта скрипта содержат число N, поместить следующие N байтов в стек
OP_1NEGATE	0x4f	Поместить значение –1 в стек
OP_RESERVED	0x50	Останов – некорректная транзакция, за исключением обнаружения в невыполняемой ветви OP_IF
OP_1 или OP_TRUE	0x51	Поместить значение 1 в стек
OP_2 – OP_16	0x52–0x60	Операция OP_N помещает значение N в стек, например OP_2 помещает 2 в стек

В табл. Б.2 описаны условные операторы, управляющие потоком выполнения.

Таблица Б.2. Условные операторы, управляющие потоком выполнения

Символ	Значение (16-ричное)	Описание
OP_NOP	0x61	Не выполняется никаких действий
OP_VER	0x62	Останов – некорректная транзакция, за исключением обнаружения в невыполняемой ветви OP_IF
OP_IF	0x63	Выполнение следующего оператора, если в вершине стека значение не 0
OP_NOTIF	0x64	Выполнение следующего оператора, если в вершине стека значение 0
OP_VERIF	0x65	Останов – некорректная транзакция
OP_VERNOTIF	0x66	Останов – некорректная транзакция
OP_ELSE	0x67	Выполнение, только если предыдущие инструкции не были выполнены
OP_ENDIF	0x68	Конец блока OP_IF, OP_NOTIF, OP_ELSE
OP_VERIFY	0x69	Проверить вершину стека, если значение не TRUE, то останов и объявление транзакции некорректной
OP_RETURN	0x6a	Останов и объявление транзакции некорректной

В табл. Б.3 описаны операторы, используемые для блокировок по времени

Таблица Б.3. Операции блокировки по времени

Символ	Значение (16-ричное)	Описание
OP_CHECKLOCKTIMEVERIFY (ранее OP_NOP2)	0xb1	Помечает транзакцию как некорректную, если значение элемента на вершине стека больше, чем значение поля транзакции nLockTime, иначе скрипт продолжает выполняться, как если бы был выполнен оператор OP_NOP. Транзакция также считается некорректной, если 1 – стек пуст, или 2 – элемент на вершине стека отрицательный, или 3 – элемент на вершине стека больше или равен 500000000, в то время как поле транзакции nLockTime меньше 500000000 или наоборот, или 4 – поле входных данных nSequence равно 0xffffffff. Более подробно смысл этого оператора описан в документе BIP-65
OP_CHECKSEQUENCEVERIFY (ранее OP_NOP3)	0xb2	Помечает транзакцию как некорректную, если относительная блокировка по времени входных данных (определяемая документом BIP 0068 с помощью параметра nSequence) не равна или больше (продолжительнее), чем значение элемента на вершине стека. Более подробно смысл этого оператора описан в документе BIP-112

В табл. Б.4 описаны операторы, используемые для работы со стеком.

Таблица Б.4. Операции со стеком

Символ	Значение (16-ричное)	Описание
OP_TOALTSTACK	0x6b	Извлекает верхний элемент из стека и помещает его в альтернативный стек
OP_FROMALTSTACK	0x6c	Извлекает верхний элемент из альтернативного стека и помещает его в стек
OP_2DROP	0x6d	Извлекает два верхних элемента стека

Окончание табл. Б.4

Символ	Значение (16-ричное)	Описание
OP_2DUP	0x6e	Дублирует два верхних элемента стека
OP_3DUP	0x6f	Дублирует три верхних элемента стека
OP_2OVER	0x70	Копирует третий и четвертый элементы в стеке на вершину
OP_2ROT	0x71	Перемещает пятый и шестой элементы в стеке на вершину
OP_2SWAP	0x72	Меняет местами две верхние пары элементов в стеке
OP_IFDUP	0x73	Дублирует верхний элемент стека, если он не равен 0
OP_DEPTH	0x74	Считает элементы в стеке и помещает значение итогового счетчика в стек
OP_DROP	0x75	Извлекает верхний элемент стека
OP_DUP	0x76	Дублирует верхний элемент стека
OP_NIP	0x77	Извлекает второй элемент стека
OP_OVER	0x78	Копирует второй элемент стека и помещает копию на вершину стека
OP_PICK	0x79	Извлекает значение N с вершины стека, затем копирует N-й элемент в вершину стека
OP_ROLL	0x7a	Извлекает значение N с вершины стека, затем перемещает N-й элемент в вершину стека
OP_ROT	0x7b	Циклически перемещает («вращает») три верхних элемента стека
OP_SWAP	0x7c	Меняет местами два верхних элемента стека
OP_TUCK	0x7d	Копирует верхний элемент стека и вставляет его между верхним и вторым элементами

В табл. Б.5 описаны операторы, используемые для работы со строками

Таблица Б.5. Операции обработки строк

Символ	Значение (16-ричное)	Описание
OP_CAT	0x7e	Запрещен (объединяет два верхних элемента стека)
OP_SUBSTR	0x7f	Запрещен (возвращает подстроку)
OP_LEFT	0x80	Запрещен (возвращает подстроку, начинающуюся с левого конца строки)
OP_RIGHT	0x81	Запрещен (возвращает подстроку, начинающуюся с правого конца строки)
OP_SIZE	0x82	Вычисляет длину строки в верхнем элементе и помещает результат в стек

В табл. Б.6 описаны операторы бинарной арифметики и булевой логики.

Таблица Б.6. Бинарная арифметика и логические условные операторы

Символ	Значение (16-ричное)	Описание
OP_INVERT	0x83	Запрещен (меняет значения битов на противоположные в верхнем элементе стека)
OP_AND	0x84	Запрещен (операция логический AND для двух верхних элементов стека)
OP_OR	0x85	Запрещен (операция логический OR для двух верхних элементов стека)

Окончание табл. Б.6

Символ	Значение (16-ричное)	Описание
OP_XOR	0x86	Запрещен (операция логический XOR для двух верхних элементов стека)
OP_EQUAL	0x87	Помещает значение TRUE (1) в стек, если два верхних элемента абсолютно равны, иначе помещает в стек значение FALSE (0)
OP_EQUALVERIFY	0x88	Аналогичен оператору OP_EQUAL, но после проверки выполняет оператор OP_VERIFY для останова, если результат не TRUE
OP_RESERVED1	0x89	Останов – некорректная транзакция, за исключением обнаружения в невыполняемой ветви OP_IF
OP_RESERVED2	0x8a	Останов – некорректная транзакция, за исключением обнаружения в невыполняемой ветви OP_IF

В табл. Б.7 описаны арифметические операторы для работы с числами.

Таблица Б.7. Операторы для работы с числами

Символ	Значение (16-ричное)	Описание
OP_1ADD	0x8b	Прибавляет 1 к значению верхнего элемента стека
OP_1SUB	0x8c	Вычитает 1 из значения верхнего элемента стека
OP_2MUL	0x8d	Запрещен (умножает значение верхнего элемента на 2)
OP_2DIV	0x8e	Запрещен (делит значение верхнего элемента на 2)
OP_NEGATE	0x8f	Меняет знак верхнего элемента стека
OP_ABS	0x90	Изменяет знак верхнего элемента на положительный (абсолютное значение)
OP_NOT	0x91	Если верхний элемент равен 0 или 1, то выполняет логическое отрицание, иначе возвращает 0
OP_0NOTEQUAL	0x92	Если верхний элемент равен 0, то возвращает 0, иначе возвращает 1
OP_ADD	0x93	Извлекает два верхних элемента, суммирует их и помещает результат в стек
OP_SUB	0x94	Извлекает два верхних элемента, вычитает первый из второго и помещает результат в стек
OP_MUL	0x95	Запрещен (умножает два верхних элемента)
OP_DIV	0x96	Запрещен (делит второй элемент на первый элемент)
OP_MOD	0x97	Запрещен (вычисляет остаток от деления второго элемента на первый элемент)
OP_LSHIFT	0x98	Запрещен (сдвигает значение второго элемента влево на количество битов, определяемое значением первого элемента)
OP_RSHIFT	0x99	Запрещен (сдвигает значение второго элемента вправо на количество битов, определяемое значением первого элемента)
OP_BOOLAND	0x9a	Операция логический AND для двух верхних элементов
OP_BOOLOR	0x9b	Операция логический OR для двух верхних элементов
OP_NUMEQUAL	0x9c	Возвращает TRUE, если два верхних элемента являются равными числами

Окончание табл. Б.7

Символ	Значение (16-ричное)	Описание
OP_NUMEQUALVERIFY	0x9d	Аналогичен NUMEQUAL, но выполняет оператор OP_VERIFY для останова, если результат не TRUE
OP_NUMNOTEQUAL	0x9e	Возвращает TRUE, если два верхних элемента не являются равными числами
OP_LESSTHAN	0x9f	Возвращает TRUE, если второй элемент меньше, чем верхний элемент
OP_GREATERTHAN	0xa0	Возвращает TRUE, если второй элемент больше, чем верхний элемент
OP_LESSTHANOREQUAL	0xa1	Возвращает TRUE, если второй элемент меньше или равен, чем верхний элемент
OP_GREATERTHANOREQUAL	0xa2	Возвращает TRUE, если второй элемент больше или равен, чем верхний элемент
OP_MIN	0xa3	Возвращает меньший из двух верхних элементов
OP_MAX	0xa4	Возвращает больший из двух верхних элементов
OP_WITHIN	0xa5	Возвращает TRUE, если третий элемент находится между вторым и первым элементами (или равен второму элементу)

В табл. Б.8 описаны операторы, реализующие криптографические функции.

Таблица Б.8. Криптографические и хэширующие операторы

Символ	Значение (16-ричное)	Описание
OP_RIPEMD160	0xa6	Возвращает хэш-значение RIPEMD160 верхнего элемента стека
OP_SHA1	0xa7	Возвращает хэш-значение SHA1 верхнего элемента стека
OP_SHA256	0xa8	Возвращает хэш-значение SHA256 верхнего элемента стека
OP_HASH160	0xa9	Возвращает хэш-значение RIPEMD160(SHA256(x)) верхнего элемента стека
OP_HASH256	0xaa	Возвращает хэш-значение SHA256(SHA256(x)) верхнего элемента стека
OP_CODESEPARATOR	0xab	Помечает начало данных, контролируемых (проверяемых) с помощью подписи
OP_CHECKSIG	0xac	Извлекает из стека открытый ключ и подпись и проверяет корректность (валидность) подписи для хэшированных данных транзакции; возвращает TRUE при соответствии
OP_CHECKSIGVERIFY	0xad	Аналог CHECKSIG, но по завершении выполняет OP_VERIFY для останова, если результат не TRUE
OP_CHECKMULTISIG	0xae	Выполняет операцию CHECKSIG для каждой переданной пары подписи и открытого ключа. Должно быть установлено соответствие всех пар. Из-за ошибки в реализации извлекается лишнее значение, поэтому для устранения этой проблемы следует использовать как префикс операцию OP_NOP
OP_CHECKMULTISIGVERIFY	0xaf	Аналог CHECKMULTISIG, но по завершении выполняет OP_VERIFY для останова, если результат не TRUE

В табл. Б.9 описаны символы, не выполняющие никаких действий.

Таблица Б.9. Символы, не выполняющие никаких действий

Символ	Значение (16-ричное)	Описание
OP_NOP1 – OP_NOP10	0xb0–0xb9	Не выполняет никаких действий, игнорируется

В табл. Б.10 описаны коды операторов, зарезервированные для использования внутренним парсером скриптов.

Таблица Б.10. Коды операторов, зарезервированные для использования внутренним парсером

Символ	Значение (16-ричное)	Описание
OP_SMALLDATA	0xf9	Представляет поле данных малого размера
OP_SMALLINTEGER	0xfa	Представляет поле малого целого числа
OP_PUBKEYS	0xfb	Представляет поля открытого ключа
OP_PUBKEYHASH	0xfd	Представляет поле хэш-значения открытого ключа
OP_PUBKEY	0xfe	Представляет поле открытого ключа
OP_INVALIDOPCODE	0xff	Представляет любой код операции, которому не назначено какое-либо действие

Приложение B

Предложения по улучшению биткойна (Bitcoin Improvement Proposals)

Предложения по улучшению биткойна (Bitcoin Improvement Proposals) – это проектные документы, предоставляющие информацию для биткойн-сообщества или содержащие описание новых функциональных возможностей биткойна, его рабочих процессов или функциональной среды.

Документ BIP-01 «BIP Purpose and Guidelines» («Цели и правила BIP») определяет три типа документов BIP:

- BIP-стандарт – описывает все изменения, которые влияют на большинство или на все реализации биткойна, например изменения в протоколе биткойна, изменения в правилах проверки (валидации) блоков или транзакций или любые изменения или дополнения, которые влияют на совместимость приложений, использующих биткойн;
- информационный BIP – описывает характеристики проектных решений биткойна или предоставляет общие правила (положения) или информацию для биткойн-сообщества, но не содержит предложений по новым функциональным возможностям. Информационные документы BIP не представляют обязательные к исполнению рекомендации или соглашения для биткойн-сообщества, поэтому пользователи и авторы реализаций по своему усмотрению могут игнорировать информационные документы BIP или следовать их рекомендациям;
- BIP с описанием процесса – описывает биткойн-процесс или цели изменения (или события) в процессе. BIP с описанием процесса похож на BIP-стандарт, но применяется в областях, не входящих непосредственно в сферу действия биткойн-протокола. Эти документы могут предла-

гать реализацию, но она не должна затрагивать основную кодовую базу биткойна. Эти документы часто требуют согласования с участием всего сообщества. В отличие от информационных BIP'ов, BIP с описанием процесса является больше, чем просто рекомендацией, и пользователи не вправе игнорировать такие документы. В качестве примеров можно назвать процедуры, правила, изменения в процессе принятия решений и изменения в инструментальных средствах или в функциональной среде, используемой в процессе разработки биткойна. Любые мета-BIP-документы также считаются BIP'ами с описанием процесса.

Документы BIP хранятся в репозитории с поддержкой различных версий на сервисе GitHub: https://github.com/bitcoin/bips. В табл. B.1 показано состояние документов BIP на апрель 2017 года. Для получения более свежей информации о существующих документах BIP и их содержимом обращайтесь в официальный репозиторий.

Таблица B.1. Состояние документов BIP на апрель 2017 года

Номер BIP	Наименование	Владелец (Автор)	Тип	Статус
BIP-1	BIP Purpose and Guidelines	Амир Тааки (Amir Taaki)	Процесс	Заменен
BIP-2	BIP process, revised	Люк Дашийр (Luke Dashjr)	Процесс	Активизирован
BIP-8	Version bits with guaranteed lock-in	Шаолин Фрай (Shaolin Fry)	Информационный	Черновик
BIP-9	Version bits with timeout and delay	Петер Вуйлле (Pieter Wuille), Питер Тодд (Peter Todd), Грег Максвелл (Greg Maxwell), Расти Расселл (Rusty Russell)	Информационный	Принят окончательно
BIP-10	Multi-Sig Transaction Distribution	Алан Райнер (Alan Reiner)	Информационный	Отменен
BIP-11	M-of-N Standard Transactions	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-12	OP_EVAL	Гэвин Андресен (Gavin Andresen)	Стандарт	Отменен
BIP-13	Address Format for pay-to-script-hash	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-14	Protocol Version and User Agent	Амир Тааки (Amir Taaki), Патрик Стрэйтмэн (Patrick Strateman)	Стандарт	Принят окончательно
BIP-15	Aliases	Амир Тааки (Amir Taaki)	Стандарт	Отложен
BIP-16	Pay to Script Hash	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-17	OP_CHECKHASHVERIFY (CHV)	Люк Дашийр (Luke Dashjr)	Стандарт	Отменен
BIP-18	hashScriptCheck	Люк Дашийр (Luke Dashjr)	Стандарт	Предложен
BIP-19	M-of-N Standard Transactions (Low SigOp)	Люк Дашийр (Luke Dashjr)	Стандарт	Черновик
BIP-20	URI Scheme	Люк Дашийр (Luke Dashjr)	Стандарт	Заменен
BIP-21	URI Scheme	Нильс Шнайдер (Nils Schneider), Мэтт Коралло (Matt Corallo)	Стандарт	Принят окончательно

Продолжение табл. В.1

Номер BIP	Наименование	Владелец (Автор)	Тип	Статус
BIP-22	getblocktemplate – Fundamentals	Люк Дашийр (Luke Dashjr)	Стандарт	Принят окончательно
BIP-23	getblocktemplate – Pooled Mining	Люк Дашийр (Luke Dashjr)	Стандарт	Принят окончательно
BIP-30	Duplicate transactions	Петер Вуйлле (Pieter Wuille)	Стандарт	Принят окончательно
BIP-31	Pong message	Майк Хёрн (Mike Hearn)	Стандарт	Принят окончательно
BIP-32	Hierarchical Deterministic Wallets	Петер Вуйлле (Pieter Wuille)	Информационный	Принят окончательно
BIP-33	Stratized Nodes	Амир Тааки (Amir Taaki)	Стандарт	Черновик
BIP-34	Block v2, Height in Coinbase	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-35	mempool message	Джеф Гарзик (Jef Garzik)	Стандарт	Принят окончательно
BIP-36	Custom Services	Стивен Томас (Stefan Thomas)	Стандарт	Черновик
BIP-37	Connection Bloom filtering	Майк Хёрн (Mike Hearn), Мэтт Коралло (Matt Corallo)	Стандарт	Принят окончательно
BIP-38	Passphrase-protected private key	Майк Колдуэлл (Mike Caldwell), Аарон Войсин (Aaron Voisine)	Стандарт	Черновик
BIP-39	Mnemonic code for generating deterministic keys	Марек Палатинус (Marek Palatinus), Павол Руснак (Pavol Rusnak), Аарон Войсин (Aaron Voisine), Шон Боу (Sean Bowe)	Стандарт	Предложен
BIP-40	Stratum wire protocol	Марек Палатинус (Marek Palatinus)	Стандарт	Номер BIP назначен
BIP-41	Stratum mining protocol	Марек Палатинус (Marek Palatinus)	Стандарт	Номер BIP назначен
BIP-42	A finite monetary supply for Bitcoin	Петер Вуйлле (Pieter Wuille)	Стандарт	Черновик
BIP-43	Purpose Field for Deterministic Wallets	Марек Палатинус (Marek Palatinus), Павол Руснак (Pavol Rusnak)	Информационный	Черновик
BIP-44	Multi-Account Hierarchy for Deterministic Wallets	Марек Палатинус (Marek Palatinus), Павол Руснак (Pavol Rusnak)	Стандарт	Предложен
BIP-45	Structure for Deterministic P2SH Multisignature Wallets	Мануэль Араос (Manuel Araoz), Райан Кс. Чарлз (Ryan X. Charles), Матиас Алехо Гарсиа (Matias Alejo Garcia)	Стандарт	Предложен
BIP-47	Reusable Payment Codes for Hierarchical Deterministic Wallets	Юстус Ранвье (Justus Ranvier)	Информационный	Черновик
BIP-49	Derivation scheme for P2WPKH-nested-in-P2SH based accounts	Дэниэл Вейгл (Daniel Weigl)	Информационный	Черновик
BIP-50	March 2013 Chain Fork Post-Mortem	Гэвин Андресен (Gavin Andresen)	Информационный	Принят окончательно

Продолжение табл. В.1

Номер BIP	Наименование	Владелец (Автор)	Тип	Статус
BIP-60	Fixed Length "version" Message (Relay-Transactions Field)	Амир Тааки (Amir Taaki)	Стандарт	Черновик
BIP-61	Reject P2P message	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-62	Dealing with malleability	Петер Вуйлле (Pieter Wuille)	Стандарт	Отменен
BIP-63	Stealth Addresses	Питер Тодд (Peter Todd)	Стандарт	Номер BIP назначен
BIP-64	getutxo message	Майк Хёрн (Mike Hearn)	Стандарт	Черновик
BIP-65	OP_CHECKLOCKTIMEVERIFY	Питер Тодд (Peter Todd)	Стандарт	Принят окончательно
BIP-66	Strict DER signatures	Петер Вуйлле (Pieter Wuille)	Стандарт	Принят окончательно
BIP-67	Deterministic Pay-to-script-hash multi-signature addresses through public key sorting	Тома Керен (Thomas Kerin), Жан-Пьер Рупп (Jean-Pierre Rupp), Рубен де Врие (Ruben de Vries)	Стандарт	Предложен
BIP-68	Relative lock-time using consensus-enforced sequence numbers	Марк Фриденбах (Mark Friedenbach), BtcDrak, Никола Дорье (Nicolas Dorier), kinoshitajona	Стандарт	Принят окончательно
BIP-69	Lexicographical Indexing of Transaction Inputs and Outputs	Кристов Атлас (Kristov Atlas)	Информационный	Предложен
BIP-70	Payment Protocol	Гэвин Андресен (Gavin Andresen), Майк Хёрн (Mike Hearn)	Стандарт	Принят окончательно
BIP-71	Payment Protocol MIME types	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-72	bitcoin: uri extensions for Payment Protocol	Гэвин Андресен (Gavin Andresen)	Стандарт	Принят окончательно
BIP-73	Use «Accept» header for response type negotiation with Payment Request URLs	Стивен Пэйр (Stephen Pair)	Стандарт	Принят окончательно
BIP-74	Allow zero value OP_RETURN in Payment Protocol	Тоби Падилья (Toby Padilla)	Стандарт	Черновик
BIP-75	Out of Band Address Exchange using Payment Protocol Encryption	Джастин Ньютон (Justin Newton), Мэтт Дэвид (Matt David), Аарон Войсин (Aaron Voisine), Джеймс МакУайт (James MacWhyte)	Стандарт	Черновик
BIP-80	Hierarchy for Non-Colored Voting Pool Deterministic Multisig Wallets	Юстус Ранвье (Justus Ranvier), Джимми Сонг (Jimmy Song)	Информационный	Отложен
BIP-81	Hierarchy for Colored Voting Pool Deterministic Multisig Wallets	Юстус Ранвье (Justus Ranvier), Джимми Сонг (Jimmy Song)	Информационный	Отложен
BIP-83	Dynamic Hierarchical Deterministic Key Trees	Эрик Ломброзо (Eric Lombrozo)	Стандарт	Черновик
BIP-90	Buried Deployments	Сухас Дафтуар (Suhas Daftuar)	Информационный	Черновик

Продолжение табл. В.1

Номер BIP	Наименование	Владелец (Автор)	Тип	Статус
BIP-99	Motivation and deployment of consensus rule changes ([soft/hard]forks)	Жорж Тимон (Jorge Timón)	Информационный	Черновик
BIP-101	Increase maximum block size	Гэвин Андресен (Gavin Andresen)	Стандарт	Отменен
BIP-102	Block size increase to 2MB	Джеф Гарзик (Jef Garzik)	Стандарт	Черновик
BIP-103	Block size following technological growth	Петер Вуйлле (Pieter Wuille)	Стандарт	Черновик
BIP-104	Block75 – Max block size like diiculty	t.khan	Стандарт	Черновик
BIP-105	Consensus based block size retargeting algorithm	BtcDrak	Стандарт	Черновик
BIP-106	Dynamically Controlled Bitcoin Block Size Max Cap	Упал Чакраборти (Upal Chakraborty)	Стандарт	Черновик
BIP-107	Dynamic limit on the block size	Вашингтон И. Санчес (Washington Y. Sanchez)	Стандарт	Черновик
BIP-109	Two million byte size limit with sigop and sighash limits	Гэвин Андресен (Gavin Andresen)	Стандарт	Отклонен
BIP-111	NODE_BLOOM service bit	Мэтт Коралло (Matt Corallo), Питер Тодд (Peter Todd)	Стандарт	Предложен
BIP-112	CHECKSEQUENCEVERIFY	BtcDrak, Марк Фриденбах (Mark Friedenbach), Эрик Ломброзо (Eric Lombrozo)	Стандарт	Принят окончательно
BIP-113	Median time-past as endpoint for lock-time calculations	Тома Керен (Thomas Kerin), Марк Фриденбах (Mark Friedenbach)	Стандарт	Принят окончательно
BIP-114	Merkelized Abstract Syntax Tree	Джонсон Ло (Johnson Lau)	Стандарт	Черновик
BIP-120	Proof of Payment	Калле Розенбаум (Kalle Rosenbaum)	Стандарт	Черновик
BIP-121	Proof of Payment URI scheme	Калле Розенбаум (Kalle Rosenbaum)	Стандарт	Черновик
BIP-122	URI scheme for Blockchain references / exploration	Марко Понтелло (Marco Pontello)	Стандарт	Черновик
BIP-123	BIP Classiication	Эрик Ломброзо (Eric Lombrozo)	Процесс	Активизирован
BIP-124	Hierarchical Deterministic Script Templates	Эрик Ломброзо (Eric Lombrozo), Уильям Суонсон (William Swanson)	Информационный	Черновик
BIP-125	Opt-in Full Replace-by-Fee Signaling	Дэвид А.Хардинг (David A. Harding), Питер Тодд (Peter Todd)	Стандарт	Предложен
BIP-126	Best Practices for Heterogeneous Input Script Transactions	Кристов Атлас (Kristov Atlas)	Информационный	Черновик
BIP-130	sendheaders message	Сухас Дафтуар (Suhas Daftuar)	Стандарт	Предложен
BIP-131	«Coalescing Transaction» Speciication (wildcard inputs)	Крис Прист (Chris Priest)	Стандарт	Черновик
BIP-132	Committee-based BIP Acceptance Process	Энди Чейз (Andy Chase)	Процесс	Отменен
BIP-133	feeilter message	Алекс Моркос (Alex Morcos)	Стандарт	Черновик

Окончание табл. В.1

Номер BIP	Наименование	Владелец (Автор)	Тип	Статус
BIP-134	Flexible Transactions	Том Цандер (Tom Zander)	Стандарт	Черновик
BIP-140	Normalized TXID	Кристиан Декер (Christian Decker)	Стандарт	Черновик
BIP-141	Segregated Witness (Consensus layer)	Эрик Ломброзо (Eric Lombrozo), Джонсон Ло (Johnson Lau), Петер Вуйлле (Pieter Wuille)	Стандарт	Черновик
BIP-142	Address Format for Segregated Witness	Джонсон Ло (Johnson Lau)	Стандарт	Отложен
BIP-143	Transaction Signature Veriica-tion for Version 0 Witness Program	Джонсон Ло (Johnson Lau), Петер Вуйлле (Pieter Wuille)	Стандарт	Черновик
BIP-144	Segregated Witness (Peer Services)	Эрик Ломброзо (Eric Lombrozo), Петер Вуйлле (Pieter Wuille)	Стандарт	Черновик
BIP-145	getblocktemplate Updates for Segregated Witness	Люк Дашийр (Luke Dashjr)	Стандарт	Черновик
BIP-146	Dealing with signature encoding malleability	Джонсон Ло (Johnson Lau), Петер Вуйлле (Pieter Wuille)	Стандарт	Черновик
BIP-147	Dealing with dummy stack element malleability	Джонсон Ло (Johnson Lau)	Стандарт	Черновик
BIP-148	Mandatory activation of segwit deployment	Шаолин Фрай (Shaolin Fry)	Стандарт	Черновик
BIP-150	Peer Authentication	Йонас Шнелли (Jonas Schnelli)	Стандарт	Черновик
BIP-151	Peer-to-Peer Communication Encryption	Йонас Шнелли (Jonas Schnelli)	Стандарт	Черновик
BIP-152	Compact Block Relay	Мэтт Коралло (Matt Corallo)	Стандарт	Черновик
BIP-171	Currency/exchange rate information API	Люк Дашийр (Luke Dashjr)	Стандарт	Черновик
BIP-180	Block size/weight fraud proof	Люк Дашийр (Luke Dashjr)	Стандарт	Черновик
BIP-199	Hashed Time-Locked Contract transactions	Шон Боу (Sean Bowe), Дайара Хопвуд (Daira Hopwood)	Стандарт	Черновик

Приложение Г

Функция Segregated Witness (Segwit)

Функция Segregated Witness (Segwit) представляет собой обновление правил консенсуса и сетевого протокола биткойн-системы, предложенное и реализованное как неустойчивое разветвление (soft fork), которое на момент написания книги (середина 2017 года) ожидает активизации.

В криптографии термин «свидетельство» (или «доказательство») (witness) используется для описания решения криптографической головоломки (задачи). В терминах биткойна свидетельство выполняет криптографическое условие, размещенное в неизрасходованных выходных данных (UTXO) транзакции.

В контексте биткойна цифровая подпись является одним из типов свидетельства, но свидетельством в более широком смысле является любое решение, которое может выполнить условия, заданные в данных UTXO, и разблокировать эти данные UTXO для расходования. Термин «свидетельство» – в этом смысле гораздо более обобщенный термин для «разблокирующего скрипта» или «scriptSig».

До появления механизма segwit каждые входные данные в транзакции сопровождались «свидетельствующими» данными, которые разблокировали эти входные данные. Свидетельствующие данные встраивались в транзакцию как часть каждых входных данных. Термин segregated witness, или segwit для краткости, означает просто подпись или разблокирующий скрипт в конкретном фрагменте выходных данных. Для простоты можно пользоваться самой простой формой этого термина: separate scriptSig или separate signature (отдельная подпись).

Таким образом, механизм Segregated Witness представляет собой архитектурное изменение в биткойн-системе, ориентированное на перемещение свидетельствующих (доказательных) данных из поля `scriptSig` (разблокирующий скрипт) транзакции в отдельную структуру данных `witness`, которая сопровождает транзакцию. Клиенты могут запросить данные транзакции вместе с сопровождающими данными witness или без них.

В этом приложении мы рассмотрим некоторые преимущества Segregated Witness, а также механизм, используемый для развертывания и реализации

этого архитектурного изменения, и продемонстрируем практическое применение Segregated Witness в транзакциях и адресах.

Механизм Segregated Witness определен в следующих документах BIP:

- BIP-141 – основное определение и описание механизма Segregated Witness;
- BIP-143 – верификация подписи транзакций (Transaction Signature Verification) для версии 0 программы свидетельства (Witness Program, version 0);
- BIP-144 – сервисы пиринговой сети (Peer Services) – новые форматы сетевых сообщений и сериализации;
- BIP-145 – обновления getblocktemplate для Segregated Witness (для выполнения майнинга).

Зачем нужен механизм Segregated Witness

Segregated Witness – это архитектурное изменение, оказывающее воздействие на масштабируемость, безопасность, финансовое стимулирование и производительность биткойн-системы, выражающееся в нескольких следующих эффектах:

- гибкость транзакций – при выносе доказательных данных witness за пределы транзакции хэш-значение транзакции используется как идентификатор, теперь уже не включаемый в witness-данные. Поскольку witness-данные являются единственной частью транзакции, которая может быть изменена третьей стороной (см. раздел «Идентификаторы транзакций» ниже в этом же приложении), удаление этой части также исключает возможность атак, использующих изменяемость (гибкость) транзакций. При использовании Segregated Witness хэш-значение транзакции становится неизменяемым для всех, кроме создателя этой транзакции, и это значительно улучшает реализации многих других протоколов, основанных на усовершенствованной конструкции биткойн-транзакций, таких как каналы платежей, цепочечные транзакции и сети типа Lightning Network;
- управление версиями скриптов – после ввода в эксплуатацию скриптов Segregated Witness в самом начале каждого блокирующего скрипта стали указывать номер версии скрипта по аналогии с транзакциями и блоками, также включающими номера версий. Добавление номера версии скрипта позволяет обновлять скриптовый язык в режиме обратной совместимости (то есть используя для обновления неустойчивые разветвления (soft fork)) при вводе новых операторов, синтаксиса и семантики скриптов. Возможность обновления скриптового языка таким безболезненным способом существенно ускоряет внедрение инноваций в биткойн;
- масштабирование сетей и хранилищ – witness-данные часто становятся причиной значительного увеличения размера транзакций. Сложные скрипты, например используемые для мультиподписей или для каналов

платежей, весьма велики. В некоторых случаях размер этих скриптов составляет даже бóльшую часть (более 75%) данных транзакции. Перемещая witness-данные за пределы транзакции, Segregated Witness улучшает масштабируемость биткойн-системы. Узлы могут отсечь witness-данные после проверки корректности подписей или вообще игнорировать их при выполнении упрощенной верификации платежа. Нет необходимости передавать witness-данные всем узлам и хранить их на дисках всех узлов;

- оптимизация верификации подписей – Segregated Witness обновляет функции, работающие с подписями (`CHECKSIG`, `CHECKMULTISIG` и т. п.), чтобы уменьшить вычислительную сложность алгоритма. До segwit алгоритм генерации подписей требовал выполнения определенного количества операций хэширования, пропорционального размеру транзакции. Сложность вычислений хэш-значений для данных увеличивается до $O(n^2)$ с учетом количества операций с подписями, создавая большую вычислительную нагрузку на все узлы, проверяющие подпись. После ввода segwit алгоритм изменился, и его сложность снизилась до $O(n)$;
- усовершенствование режима создания подписи в режиме офлайн – подписи Segregated Witness включают значение (сумму), на которую ссылается каждый фрагмент входных данных, заверенный такой подписью. Раньше устройство создания подписи в режиме офлайн, например аппаратный кошелек, обязательно должно было проверить сумму, указанную в каждом фрагменте входных данных, перед подписью транзакции. Обычно эта операция выполнялась с передачей большого объема данных о предыдущей транзакции, являющейся источником обрабатываемых входных данных. Но после ввода segwit этот объем стал частью передаваемого хэш-значения, которое нужно заверить подписью, поэтому офлайновому устройству не нужна предыдущая транзакция. Если суммы не совпадают (искажены скомпрометированной онлайновой системой), то подпись считается некорректной.

Как работает механизм Segregated Witness

На первый взгляд, Segregated Witness выглядит как изменение в процессе создания транзакций, следовательно, как новая функциональная возможность на уровне транзакций, но это не так. В действительности Segregated Witness также изменяет процедуру расходования отдельных данных UTXO, то есть представляет собой функцию, действующую на уровне выходных данных.

Транзакция может расходовать выходные данные, использующие Segregated Witness, или обычные (содержащие witness-данные) выходные данные, или оба типа выходных данных. Таким образом, нет какого-либо смысла обозначать транзакцию специальным термином «Segregated Witness транзакция». Вместо этого следует особо определить «выходные данные с использованием Segregated Witness».

Когда транзакция расходует данные UTXO, она обязана предоставить доказательство (witness). В обычных данных UTXO блокирующий скрипт требует, чтобы witness-данные были включены в выходные данные транзакции, которая расходует эти данные UTXO. Но данные UTXO, использующие Segregated Witness, определяют блокирующий скрипт, условия которого могут быть выполнены с помощью witness-данных, размещенных вне входных данных (отделенных от них).

Неустойчивое разветвление (обратная совместимость)

Segregated Witness представляет собой существенное изменение в формировании архитектуры выходных данных и транзакций. Подобное изменение обычно требует одновременного внесения корректировок на каждом биткойн-узле и в каждом кошельке для редактирования правил консенсуса, а эта процедура известна под названием устойчивое разветвление (hard fork).

Вместо этого Segregated Witness вводится с гораздо менее разрушительными изменениями, обеспечивая обратную совместимость, что определяется как неустойчивое разветвление (soft fork). Этот тип обновления позволяет необновившемуся программному обеспечению просто игнорировать внесенные изменения и продолжать работу без каких-либо затруднений и потерь.

Выходные данные Segregated Witness формируются таким образом, чтобы более старые системы, которые не используют segwit, имели возможность их проверить. Для старой версии кошелька или узла выходные данные Segregated Witness выглядят как выходные данные, которые любой может израсходовать. Такие выходные данные могут расходоваться с пустым полем подписи, следовательно, тот факт, что в транзакции отсутствует подпись (она размещена отдельно), не делает транзакцию некорректной. Но более новые версии кошельков и узлов майнинга видят выходные данные Segregated Witness, и предполагается, что они находят корректное доказательство для транзакции в ее witness-данных.

Примеры использования выходных данных Segregated Witness в транзакциях

Рассмотрим несколько примеров транзакций и понаблюдаем, как Segregated Witness воздействует на процедуры обработки транзакций. В первом примере показано, как изменяется платеж Pay-to-Public-Key-Hash (P2PKH) при использовании программного обеспечения Segregated Witness. Затем демонстрируется аналогичный пример применения Segregated Witness для скриптов Pay-to-Script-Hash (P2SH). В последнем примере рассматривается возможность встраивания обеих вышеупомянутых программ Segregated Witness в P2SH-скрипт.

Платеж Pay-to-Witness-Public-Key-Hash (P2WPKH)

В разделе «Покупка чашки кофе» главы 1 Алиса создала транзакцию для платежа Бобу за чашку кофе. Эта транзакция формирует выходные данные P2PKH со значением 0.015BTC, которое стало доступно для расходования Бобом. Скрипт выходных данных выглядит следующим образом:

```
DUP HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 EQUALVERIFY CHECKSIG
```

С помощью механизма Segregated Witness Алиса могла бы создать скрипт Pay-to-Witness-Public-Key-Hash (P2WPKH) следующего вида:

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Здесь можно видеть, что блокирующий скрипт для выходных данных Segregated Witness намного проще, чем скрипт для обычных выходных данных. Он состоит из двух значений, которые помещаются в стек выполнения скрипта. Для старых версий (не поддерживающих segwit) биткойн-клиента эти два значения в стеке должны выглядеть как выходные данные, которые может расходовать кто угодно и которые не требуют подписи (или могут расходоваться с пустым полем подписи). Для новых версий (с поддержкой segwit) биткойн-клиента первое число (0) интерпретируется как номер версии (версия доказательства – witness version), а вторая часть (20 байтов) представляет собой эквивалент блокирующего скрипта, называемого программой доказательства (witness program). 20-байтовая программа доказательства – это просто хэш-значение открытого ключа, точно такое же, как в скрипте P2PKH.

Теперь рассмотрим соответствующую транзакцию, которую Боб использует для расходования этих выходных данных. В первоначальную версию скрипта (без segwit) транзакция Боба должна была включить подпись в состав входных данных транзакции:

```
[...]
"Vin" : [
"txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
"vout": 0,
       "scriptSig": "<Bob's scriptSig>",
]
[...]
```

Но для расходования выходных данных Segregated Witness во входных данных этой транзакции подписи нет. Вместо этого транзакция Боба содержит пустое поле scriptSig и включает данные Segregated Witness вне пределов самой транзакции:

```
[...]
"Vin" : [
"txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
"vout": 0,
       "scriptSig": "",
]
```

```
[...]
"witness": "<Bob's witness data>"
[...]
```

Создание скрипта P2WPKH в кошельке

Очень важно отметить, что скрипт P2WPKH должен создаваться только получателем платежа, а не формироваться отправителем на основе известного открытого ключа, P2PKH-скрипта или адреса. У отправителя нет возможности узнать, имеется ли у кошелька получателя возможность создания segwit-транзакций и расходования выходных данных P2WPKH.

Кроме того, выходные данные P2WPKH обязательно должны создаваться на основе хэш-значения сжатого открытого ключа. Открытые ключи без сжатия считаются нестандартными в segwit и могут быть в явной форме запрещены в будущем неустойчивом разветвлении (soft fork). Если хэш-значение, используемое в скрипте P2WPKH, вычислено по несжатому открытому ключу, то оно может быть нерасходуемым, и вы потеряете свои деньги. Выходные данные P2WPKH должны создаваться кошельком получателя платежа с помощью генерации сжатого открытого ключа из секретного ключа получателя.

Скрипт P2WPKH должен создаваться получателем платежа посредством преобразования сжатого открытого ключа в хэш-значение P2WPKH. Никогда не следует выполнять преобразование скрипта P2PKH, биткойн-адреса или несжатого открытого ключа в witness-скрипт P2WPKH.

Скрипт Pay-to-Witness-Script-Hash (P2WSH)

Второй тип witness-программы связан со скриптом Pay-to-Script-Hash (P2SH), который мы рассматривали в разделе «Скрипт Pay-to-Script-Hash (P2SH)» главы 7. В примере из этого раздела скрипт P2SH использовался компанией Мохаммеда для формирования скрипта мультиподписи. Платежи, направляемые в компанию Мохаммеда, кодировались с помощью следующего блокирующего скрипта:

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

Скрипт P2SH ссылается на хэш-значение погашающего скрипта (redeem script), который определяет требование мультиподписи 2-of-3 для расходования денежных средств. Для расходования этих выходных данных компания Мохаммеда должна представить погашающий скрипт (хэш-значение которого совпадает с хэш-значением скрипта в выходных данных P2SH) и соответствующие подписи, необходимые для выполнения условий этого погашающего скрипта, – все эти данные должны быть включены во входные данные транзакции:

```
[...]
"Vin" : [
"txid": "abcdef12345...",
"vout": 0,
       "scriptSig": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 CHECKMULTISIG>",
]
```

Теперь рассмотрим, как можно обновить этот пример для применения segwit. Если бы клиенты Мохаммеда пользовались кошельками, совместимыми с segwit, то они могли бы выполнить платеж, создав выходные данные Pay-to-Witness-Script-Hash (P2WSH) следующего вида:

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Здесь, как и в примере с P2WPKH, можно видеть, что аналогичный скрипт Segregated Witness намного проще, в нем нет различных операторов, которые содержались в скриптах P2SH. Вместо этого программа Segregated Witness состоит из двух значений, помещенных в стек: версия доказательства witness (0) и 32-байтовое хэш-значение SHA256 погашающего скрипта (redeem script).

Скрипт P2SH использует 20-байтовое хэш-значение `RIPEMD160(SHA256(script))`, в то время как witness-программа P2WSH использует 32-байтовое хэш-значение `SHA256(script)`. Это различие в выборе алгоритма хэширования является вполне взвешенным решением, которое позволяет различать два типа witness-программ (P2WPKH и P2WSH) по длине хэш-значения, а также обеспечивает более прочную защиту P2WSH (128 битов, по сравнению с 80 битами P2SH).

Компания Мохаммеда может расходовать фрагменты выходных данных P2WSH, предоставляя корректный погашающий скрипт и достаточное количество подписей для удовлетворения его условий. И погашающий скрипт, и подписи должны быть отделены от расходующей транзакции в виде части witness-данных. Во входных данных этой транзакции кошелек Мохаммеда должен разместить пустое поле scriptSig:

```
[...]
"Vin" : [
"txid": "abcdef12345...",
"vout": 0,
       "scriptSig": "",
]
[...]
"witness": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5 CHECKMULTISIG>"
[...]
```

Различия между P2WPKH и P2WSH

В двух предыдущих разделах мы рассматривали два типа witness-программ: Pay-to-Witness-Public-Key-Hash (P2WPKH) и Pay-to-Witness-Script-Hash (P2WSH). Оба типа witness-программ состоят из одного байта номера версии, за которым следует более длинное хэш-значение. Они очень похожи, но интерпретируются совершенно по-разному: одно интерпретируется как хэш-значение открытого ключа, для которого должна быть представлена соответствующая подпись, другое интерпретируется как хэш-значение скрипта, условия которого выполняются с помощью погашающего скрипта. Главным различием между этими типами является длина хэш-значения:

- хэш-значение открытого ключа в P2WPKH имеет длину 20 байтов;
- хэш-значение скрипта в P2WSH имеет длину 32 байта.

Это единственное различие, которое позволяет кошельку отличить один тип witness-программы от другого. Оценивая длину хэш-значения, кошелек может определить, какой тип witness-программы применяется в каждом конкретном случае – P2WPKH или P2WSH.

Обновление ПО для использования Segregated Witness

В предыдущих примерах можно видеть, что обновление ПО с целью использования Segregated Witness представляет собой двухэтапный процесс. На первом этапе кошельки должны создавать выходные данные особого типа segwit. Затем эти выходные данные могут расходоваться кошельками, которым известно, как формировать транзакции типа Segregated Witness. В этих примерах кошелек Алисы считался segwit-совместимым и способным создавать специализированные выходные данные с помощью скриптов Segregated Witness. Было принято, что кошелек Боба тоже segwit-совместим и может расходовать эти выходные данные. Возможно, не вполне очевидным в этих примерах было то, что на практике кошелек Алисы должен знать, что Боб использует segwit-совместимый кошелек и способен расходовать выходные данные Segregated Witness. Если же кошелек Боба не обновлен, а Алиса пытается выполнить segwit-платежи Бобу, то кошелек Боба не сможет обнаружить этих платежей.

При платежах типа P2WPKH и P2WSH кошельки отправителя и получателя необходимо обновить, чтобы они могли воспользоваться механизмом segwit. Более того, кошелек отправителя должен знать, что кошелек получателя является segwit-совместимым.

Segregated Witness не будет одновременно реализован во всей сети в целом. Вместо этого реализация Segregated Witness выполняется с соблюдением обратной совместимости, то есть старые и новые версии ПО клиентов могут работать вместе. Разработчики кошельков будут независимо друг от друга обновлять программное обеспечение, добавляя в него функциональные возможности segwit. Платежи типа P2WPKH и P2WSH используются, когда отправитель и получатель являются segwit-совместимыми. Обычные платежи P2PKH и P2SH будут продолжать работать для необновленных кошельков. При этом возможны два важных сценария, которые рассматриваются в следующем разделе:

- возможность segwit-несовместимого кошелька отправителя выполнить платеж кошельку получателя, который способен обрабатывать segwit-транзакции;
- возможность segwit-совместимого кошелька отправителя распознавать и отличать от прочих получателей segwit-совместимые кошельки по их адресам.

Включение данных Segregated Witness в скрипт P2SH

Предположим, например, что кошелек Алисы не обновлен до совместимости с segwit, а кошелек Боба обновлен и может обрабатывать segwit-транзакции. Алиса и Боб могут использовать «старую» транзакцию без segwit. Но, вероятнее

всего, Боб хотел бы воспользоваться механизмом segwit для снижения оплаты за транзакции, то есть предоставлением скидок, применяемых для witness-данных.

В этом случае кошелек Боба может создать адрес P2SH, содержащий внутри скрипт segwit. Кошелек Алисы видит «обычный» адрес P2SH и может выполнять на него платежи без каких-либо знаний о segwit. Затем кошелек Боба может расходовать эти платежи в segwit-транзакции, используя все преимущества механизма segwit и снижая плату за транзакции.

Обе формы witness-скриптов, P2WPKH и P2WSH, могут быть включены (встроены) в адрес P2SH. Первая форма обозначается P2SH(P2WPKH), вторая – P2SH(P2WSH).

Скрипт Pay-to-Witness-Public-Key-Hash, включенный в скрипт Pay-to-Script-Hash

Сначала рассмотрим первую форму witness-скрипта P2SH(P2WPKH). Это программа Pay-to-Witness-Public-Key-Hash, включенная (встроенная) в скрипт Pay-to-Script-Hash, которая может использоваться кошельком, незнакомым с механизмом segwit.

Кошелек Боба формирует witness-программу P2WPKH с помощью открытого ключа Боба. Затем эта witness-программа хэшируется и полученное хэш-значение кодируется как скрипт P2SH. Скрипт P2SH преобразуется в биткойн-адрес, который начинается с цифры «3», как мы могли видеть в разделе «Скрипт Pay-to-Script-Hash (P2SH)» в главе 7.

Кошелек Боба начинает операцию с witness-программой P2WPKH, которую мы видели ранее, в предыдущих примерах:

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Эта witness-программа P2WPKH состоит из номера версии доказательства (свидетельства) и 20-байтового хэш-значения открытого ключа Боба.

Затем кошелек Боба хэширует показанную выше witness-программу, сначала с помощью функции SHA256, потом с помощью функции RIPEMD160, получая другое 20-байтовое хэш-значение:

```
3e0547268b3b19288b3adef9719ec8659f4b2b0b
```

После этого хэш-значение witness-программы включается в скрипт P2SH:

```
HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL
```

В конце выполняется преобразование скрипта P2SH в биткойн-адрес P2SH:

```
37Lx99uaGn5avKBxiW26HjedQE3LrDCZru
```

Теперь Боб может предъявить этот адрес покупателям для оплаты приобретаемых порций кофе. Кошелек Алисы может выполнить платеж на адрес 3deadbeef точно так же, как платил бы на любой другой биткойн-адрес. Даже если кошелек Алисы не поддерживает segwit, созданный им платеж может быть израсходован Бобом в segwit-транзакции.

Скрипт Pay-to-Witness-Script-Hash, включенный в скрипт Pay-to-Script-Hash

Аналогичным образом witness-программа P2WSH для скрипта мультиподписи или любой другой сложный скрипт может быть включен (встроен) в скрипт и адрес P2SH, предоставляя возможность любому другому кошельку выполнять segwit-совместимые платежи.

Как мы видели выше в разделе «Скрипт Pay-to-Witness-Script-Hash (P2WSH)» этого приложения, компания Мохаммеда использует платежи Segregated Witness, направляемые скриптам с мультиподписями. Чтобы любой клиент имел возможность заплатить компании, вне зависимости от того, совместим ли его кошелек с механизмом segwit или нет, кошелек Мохаммеда может включить witness-программу P2WSH в скрипт P2SH.

Сначала кошелек Мохаммеда создает witness-программу P2WSH, соответствующую скрипту с мультиподписями, хэшированному с помощью функции SHA256:

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Затем сама witness-программа хэшируется дважды с помощью функций SHA256 и RIPEMD160, и в результате получается новое 20-байтовое хэш-значение, аналогичное используемому в обычном скрипте P2SH:

```
86762607e8fe87c0c37740cddee880988b9455b2
```

Далее кошелек Мохаммеда помещает вычисленное хэш-значение в скрипт P2SH:

```
HASH160 86762607e8fe87c0c37740cddee880988b9455b2 EQUAL
```

Наконец, кошелек генерирует биткойн-адрес из этого скрипта:

```
3Dwz1MXhM6EfFoJChHCxh1jWHb8GqqRenG
```

Теперь клиенты Мохаммеда могут выполнять платежи на этот адрес без обязательной поддержки механизма segwit их кошельками. Компания Мохаммеда может формировать segwit-транзакции для расходования этих платежей, пользуясь всеми преимуществами segwit, в том числе и более низкой платой за транзакции.

Адреса Segregated Witness

После развертывания segwit в биткойн-сети потребуется некоторое время на обновление кошельков. Поэтому, вероятнее всего, segwit будет главным образом использоваться в форме встраивания в скрипты P2SH, как мы видели в предыдущем разделе, по крайней мере в течение нескольких месяцев.

Но в конечном итоге практически все кошельки перейдут к поддержке segwit-платежей. К тому времени исчезнет необходимость встраивания segwit в скрипты P2SH. Следовательно, почти наверняка будет создана новая фор-

ма биткойн-адреса, по которой отправитель узнает о поддержке механизма segwit и будет напрямую кодировать witness-программу. Уже внесено несколько предложений по схеме адреса Segregated Witness, но пока ни одно из них не обсуждается достаточно активно.

Идентификаторы транзакций

Одним из самых главных преимуществ механизма Segregated Witness является то, что он устраняет возможность легкого изменения транзакций третьей стороной.

До segwit транзакции могли содержать соответствующие подписи, слегка подредактированные третьими сторонами, изменяющими идентификатор (ID) транзакции (хэш-значение) без внесения изменений в основные свойства (входные и выходные данные, суммы). Это создавало возможности для проведения атак типа DoS, а также атак, использующих некачественно написанное программное обеспечение кошельков, которое принимало хэш-значения неподтвержденных транзакций как неизменяемое.

После ввода в эксплуатацию механизма Segregated Witness транзакции стали получать два идентификатора: `txid` и `wtxid`. Обычный идентификатор транзакции `txid` представляет собой результат двойного хэширования функцией SHA256 сериализованной транзакции без witness-данных. Идентификатор транзакции `wtxid` – результат двойного хэширования функцией SHA256 нового формата сериализации транзакции с witness-данными.

Обычный идентификатор `txid` вычисляется точно так же, как для транзакции без использования segwit. Но поскольку segwit-транзакция содержит пустое поле `scriptSig` в каждом фрагменте входных данных, здесь нет частей транзакции, которые могут быть изменены третьей стороной. Таким образом, в segwit-транзакции идентификатор `txid` является неизменяемым для любой третьей стороны, даже если эта транзакция не подтверждена.

Идентификатор `wtxid` в какой-то степени является «расширенным» (дополнительным) идентификатором, в хэш-значение которого включены также witness-данные. Если транзакция передается без witness-данных, то значения `wtxid` и `txid` одинаковы. Отметим, что поскольку `wtxid` включает witness-данные (подписи) и поскольку witness-данные могут быть легко изменяемыми, сам идентификатор `wtxid` также должен считаться легко изменяемым до тех пор, пока транзакция не будет подтверждена. Только идентификатор `txid` segwit-транзакции можно считать неизменяемым третьими сторонами, и только если все фрагменты входных данных транзакции являются входными данными Segregated Witness.

Транзакции Segregated Witness имеют два идентификатора: `txid` и `wtxid`. Идентификатор `txid` – это хэш-значение транзакции без witness-данных. Идентификатор `wtxid` – это хэш-значение транзакции с включением witness-данных. Идентификатор `txid` транзакции, в которой все фрагменты входных данных представляют собой входные данные Segregated Witness, не подвержен каким-либо изменениям третьих сторон.

Новый алгоритм подписи в механизме Segregated Witness

Механизм Segregated Witness изменяет семантику четырех функций проверки подписи (CHECKSIG, CHECKSIGVERIFY, CHECKMULTISIG и CHECKMULTISIGVERIFY), так как изменяет способ вычисления хэш-значения передаваемой транзакции.

Подписи в биткойн-транзакциях применяются к передаваемому хэш-значению, которое вычисляется на основе данных транзакции с блокировкой определенных частей этих данных, означающей принятие подписывающей стороной обязательств по отношению к этим данным. Например, в простом типе подписи SIGHASH_ALL хэш-значение передаваемого обязательства включает все входные и выходные данные.

К сожалению, существовавший до сих пор способ вычисления хэш-значения передаваемого обязательства вынуждал узел, проверяющий подпись, выполнять значительный объем хэш-вычислений. В частности, сложность хэш-операций увеличивается до $O(n^2)$ с учетом количества операций подписи в транзакции. Таким образом, атакующий мог создать транзакцию с огромным количеством операций подписи, тем самым заставляя всю биткойн-сеть выполнять сотни тысяч хэш-операций по верификации этой транзакции.

Механизм segwit предоставляет возможность устранения этой проблемы, изменяя способ вычисления хэш-значения передаваемого обязательства. Для witness-программ segwit версии 0 верификация подписей выполняется с использованием усовершенствованного алгоритма хэширования передаваемого обязательства, определенного и описанного в документе BIP-143.

Новый алгоритм служит достижению двух важных целей. Во-первых, увеличение количества хэш-операций находится в более мягкой, линейной зависимости (сложность O(n)) от количества операций подписи, тем самым снижая вероятность проведения атак типа DoS с помощью чрезмерно усложненных транзакций. Во-вторых, хэш-значение передаваемого обязательства теперь включает еще и значения (суммы) каждого фрагмента входных данных, то есть устраняет необходимость «вытягивания» и проверки данных из предыдущей транзакции, на которую ссылаются текущие входные данные. При использовании офлайновых устройств, таких как аппаратные кошельки, это существенно упрощает обмен информацией между хостом и аппаратным кошельком, полностью исключая необходимость передачи предыдущих транзакций для валидации. Аппаратный кошелек может принять значение входных данных «как указано» от ненадежного в плане доверия хоста. Поскольку подпись недействительна, если это значение входных данных некорректно, аппаратному кошельку нет нужды проверять корректность значения перед подписью входных данных.

Экономические стимулы для использования механизма Segregated Witness

Узлы майнинга биткойнов и полноценные узлы принимают на себя затраты по оплате ресурсов, используемых для поддержки биткойн-сети и структуры

данных блокчейна. С увеличением объема биткойн-транзакций увеличиваются и затраты ресурсов (CPU, пропускная способность сети, дисковое пространство, память). Майнеры получают компенсацию за эти затраты в виде отчислений, пропорциональных размеру (в байтах) каждой транзакции. Полноценные узлы, не занимающиеся майнингом, компенсации не получают, поэтому принимают на себя затраты из-за необходимости поддержки работы полноценного надежного узла с функциями полной проверки (валидации) всей базы данных, возможно, потому что такой узел используется для биткойн-бизнеса.

Без оплаты транзакций рост объема биткойн-данных, вероятнее всего, был бы весьма существенным. Оплата транзакций направлена на регулирование потребностей пользователей биткойн-системы, нагружающих своими транзакциями сеть. Это регулирование осуществляется с помощью механизма определения цен по рыночным стоимостям.

При вычислении оплаты на основе размера транзакций для всех данных в транзакции устанавливается одинаковая стоимость. Но с точки зрения полноценных узлов и майнеров стоимость некоторых частей транзакций следует оценивать гораздо более высоко. Каждая транзакция, добавляемая в биткойн-сеть, влияет на потребление четырех ресурсов узлов:

- дисковое пространство – каждая транзакция записывается в структуру данных блокчейна, увеличивая общий размер этой структуры. Структура данных блокчейна хранится на жестком диске, но хранение можно оптимизировать, «отсекая» самые старые транзакции;
- CPU – каждая транзакция требует обязательной валидации (проверки корректности), на это требуются затраты процессорного времени;
- пропускная способность – каждая транзакция передается (по методике лавинного распространения информации) по сети как минимум однократно. Без какой-либо оптимизации протокола распространения блоков транзакции передаются повторно как части блока, увеличивая вдвое нагрузку на пропускную способность сети;
- оперативная память – узлы, выполняющие валидацию транзакций, хранят индекс данных UTXO или полный набор данных UTXO в оперативной памяти для ускорения процесса проверки. Поскольку оперативная память как минимум на порядок дороже внешней памяти на дисках, рост объема данных UTXO вносит заметную диспропорцию в накладные расходы по сопровождению функционирующего узла.

Как можно видеть из этого списка, не все части транзакций в одинаковой мере воздействуют на стоимость поддержки работы узла или на возможность масштабирования биткойн-системы для поддержки большего объема транзакций. Самой затратной частью транзакций являются вновь создаваемые выходные данные, так как они добавляются в набор данных UTXO, хранимый в оперативной памяти. Для сравнения: подписи (то есть witness-данные) создают меньшую нагрузку на сеть и меньшие накладные расходы на сопровождение узлов, так как witness-данные проверяются только один раз и после проверки

никогда не используются. Более того, сразу после получения новой транзакции и процедуры валидации witness-данных узлы могут удалить эти witness-данные. Если оплата транзакций вычисляется по их размеру без предпочтения одного из этих двух типов данных, то рыночные стимулы в виде отчислений не регулируются с помощью реальных стоимостей (затрат), определяемых конкретной транзакцией. В действительности существующая ныне структура отчислений за транзакции поощряет противоположное поведение, потому что witness-данные являются самой большой частью транзакции.

Стимулы, создаваемые отчислениями за транзакции, имеют большое значение, так как они влияют на поведение кошельков. Все кошельки обязаны реализовать некоторую стратегию формирования (сборки) транзакций, которая учитывает ряд факторов, таких как приватность (сокращение и даже устранение повторного использования адреса), фрагментация (огромное количество беспорядочных изменений) и оплата транзакций. Если отчисления за транзакции чрезмерно мотивируют кошельки использовать столько фрагментов входных данных, сколько возможно в транзакциях, то такой подход может привести к дроблению данных UTXO и изменению стратегий создания адресов и, как следствие, к непредвиденному огромному росту общего объема данных UTXO.

Транзакции используют данные UTXO в своих входных данных и создают новые данные UTXO в выходных данных. Таким образом, транзакция, в которой больше фрагментов входных данных, чем фрагментов выходных данных, способствует уменьшению размера набора данных UTXO, тогда как транзакция с бóльшим количеством фрагментов выходных данных, по сравнению с количеством фрагментов входных данных, способствует увеличению объема данных UTXO. Рассмотрим подробнее это различие между количеством фрагментов входных и выходных данных и назовем его «итоговое число новых данных UTXO» (Net-new-UTXO). Это важная метрика, поскольку она дает информацию о степени воздействия транзакции на самый дорогостоящий ресурс во всей сети – хранимый в оперативной памяти набор данных UTXO. Транзакция с положительным итоговым числом новых данных UTXO увеличивает нагрузку на этот ресурс. Транзакция с отрицательным итоговым числом новых данных UTXO снижает нагрузку. Следовательно, необходимо поощрять транзакции с отрицательным итоговым числом новых данных UTXO или хотя бы с нейтральным нулевым итоговым числом новых данных UTXO.

Рассмотрим на примере, какие стимулы создаются при вычислении оплаты за транзакции с использованием и без использования механизма Segregated Witness. Возьмем две различные транзакции. Транзакция А содержит 3 фрагмента входных данных и 2 фрагмента выходных данных, то есть имеет метрику «итоговое число новых данных UTXO» –1, потребляя на один фрагмент данных UTXO больше, чем создает, что приводит к уменьшению общего размера данных UTXO на единицу. Транзакция Б содержит 2 фрагмента входных данных и 3 фрагмента выходных данных, метрика «итоговое число новых данных UTXO» равна 1, то есть в набор данных UTXO добавляется один но-

вый фрагмент и создаются дополнительные затраты во всей биткойн-сети. Обе транзакции используют скрипты с мультиподписями (2-of-3), чтобы показать, как сложные скрипты усиливают воздействие механизма Segregated Witness на определение оплаты транзакций. Предположим, что оплата составляет 30 сатоши за байт, а также установлена 75-процентная скидка на witness-данные:

Без использования механизма Segregated Witness
Оплата транзакции А: 25 710 сатоши
Оплата транзакции Б: 18 990 сатоши

С использованием механизма Segregated Witness
Оплата транзакции А: 8130 сатоши
Оплата транзакции Б: 12 045 сатоши

При использовании реализации механизма Segregated Witness обе транзакции становятся дешевле. Но при сравнении стоимостей обеих транзакций между собой мы видим, что без механизма Segregated Witness более высокая оплата берется за транзакцию с отрицательной метрикой «итоговое число новых данных UTXO». При использовании механизма Segregated Witness сумма оплаты транзакций выравнивается с поощрением минимизации количества новых создаваемых данных UTXO при отсутствии непреднамеренных наказаний транзакций с большим количеством входных данных.

Таким образом, механизм Segregated Witness создает два основных эффекта воздействия на определение платы за транзакции, вносимой пользователями биткойн-системы. Во-первых, segwit сокращает общую стоимость транзакций с помощью скидки за witness-данные и увеличивает емкость структуры данных блокчейна биткойн-системы. Во-вторых, принятая в segwit скидка на witness-данные сглаживает несбалансированность стимулов, которая могла бы стать косвенной причиной чрезмерного роста объема данных UTXO.

Приложение Д

Bitcore

Bitcore – это комплект инструментальных средств, предоставляемых BitPay. Его главная цель – предоставить разработчикам биткойна простые и удобные инструменты. Почти весь код Bitcore написан на языке JavaScript. Существует несколько модулей, специально написанных для NodeJS. Модуль node в последней версии Bitcore включает код на языке C++ из Bitcoin Core. Более подробную информацию см. на сайте https://bitcore.io.

Список функциональных возможностей Bitcore

- Полноценный узел биткойн-системы (bitcore-node).
- Проводник блоков (insight).
- Утилиты для работы с блоками, транзакциями и кошельками (bitcore-lib).
- Поддержка прямого обмена данными с пиринговой биткойн-сетью (bitcore-p2p).
- Генерация мнемонического источника энтропии (bitcore-mnemonic).
- Протокол платежа (bitcore-payment-protocol).
- Верификация и подпись сообщений (bitcore-message).
- Реализация схемы комплексного шифрования с использованием эллиптических кривых (bitcore-ecies).
- Сервис кошелька (bitcore-wallet-service).
- Клиент-кошелек (bitcore-wallet-client).
- «Песочница» (bitcore-playground).
- Объединенные сервисы, взаимодействующие напрямую с Bitcoin Core (bitcore-node).

Примеры использования библиотеки Bitcore

Предварительные сведения

- Для выполнения примеров требуется версия NodeJS >= 4.x, или необходимо воспользоваться управляемой онлайновой «песочницей» (https://bitcore.io/playground).

При использовании NodeJS и цикла «чтение–вычисление–вывод» (REPL) узла:

```
$ npm install -g bitcore-lib bitcore-p2p
$ NODE_PATH=$(npm list -g | head -1)/node_modules node
```

Примеры кошелька, использующего bitcore-lib

Создание нового биткойн-адреса с соответствующим секретным ключом:

```
> bitcore = require('bitcore-lib')
> privateKey = new bitcore.PrivateKey()
> address = privateKey.toAddress().toString()
```

Создание иерархического детерминированного секретного ключа и адреса:

```
> hdPrivateKey = bitcore.HDPrivateKey()
> hdPublicKey = bitcore.HDPublicKey(hdPrivateKey)
> hdAddress = new bitcore.Address(hdPublicKey.publicKey).toString()
```

Создание и подпись транзакции из данных UTXO:

```
> utxo = {
  txId: transaction id containing an unspent output,
  outputIndex: output indexi e.g. 0,
  address: addressOfUtxo,
  script: bitcore.Script.buildPublicKeyHashOut(addressOfUtxo).toString(),
  satoshis: amount sent to the address
}
> fee = 3000 //set appropriately for conditions on the network
> tx = new bitcore.Transaction()
       .from(utxo)
       .to(address, 35000)
       .fee(fee)
       .enableRBF()
       .sign(privateKeyOfUtxo)
```

Замена самой последней транзакции в пуле памяти (replace-by-fee):

```
> rbfTx = new Transaction()
          .from(utxo)
          .to(address, 35000)
          .fee(fee*2)
          .enableRBF()
          .sign(privateKeyOfUtxo);
> tx.serialize();
> rbfTx.serialize();
```

Распространение транзакции в биткойн-сети (примечание: распространяются только корректные транзакции; для получения информации о пиринговых хостах см. https://bitnodes.21.co/nodes):

1. Скопировать приведенный ниже исходный код в файл с именем *broadcast.js*.

2. Переменные tx и rbfTx обозначают выходные данные функций tx.serialize() и rbfTx.serialize() соответственно.
3. Для выполнения операции замены по схеме replace-by-fee партнер обязательно должен обеспечить поддержку ключа mempoolreplace программы bitcoind и установить его значение равным 1.
4. Выполнить файл для запуска узла *broadcast.js*:

```
var p2p = require('bitcore-p2p');
var bitcore = require('bitcore-lib');
var tx = new bitcore.Transaction('output from serialize function');
var rbfTx = new bitcore.Transaction('output from serialize function');
var host = 'ip address'; //use valid peer listening on tcp 8333
var peer = new p2p.Peer({host: host});
var messages = new p2p.Messages();
peer.on('ready', function() {
  var txs = [messages.Transaction(tx), messages.Transaction(rbfTx)];
  var index = 0;
  var interval = setInterval(function() {
    peer.sendMessage(txs[index++]);
    console.log('tx: ' + index + ' sent');
    if (index === txs.length) {
      clearInterval(interval);
      console.log('disconnecting from peer: ' + host);
      peer.disconnect();
    }
  }, 2000);
});
peer.connect();
```

Приложение E

Библиотека pycoin, утилиты ku и tx

Библиотека на языке Python pycoin (http://github.com/richardkiss/pycoin), начальная версия которой была написана и сопровождалась Ричардом Киссом (Richard Kiss), представляет собой библиотеку языка Python, поддерживающую работу с ключами и транзакциями в биткойн-системе. Кроме того, библиотека обеспечивает поддержку скриптового языка, достаточную для корректной обработки нестандартных транзакций.

Библиотека pycoin поддерживает обе текущие версии языка: Python 2 (2.7.x) и Python 3 (3.3 и более поздние версии). В состав библиотеки включены весьма полезные утилиты командной строки: ku и tx.

Утилита для работы с ключами ku (Key Utility)

Утилита командной строки ku (key utility) представляет собой своеобразный «швейцарский армейский универсальный нож» для работы с ключами. Она поддерживает ключи по стандарту BIP-32, WIF и адреса (для биткойнов и других криптовалют). Ниже приводятся практические примеры использования этой утилиты.

Создание ключа по стандарту BIP-32 с использованием принятых по умолчанию источников энтропии из GPG и */dev/random*:

```
$ ku create
input           : create
network         : Bitcoin
wallet key      : xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5Su9DcWHvXJe-
miJBsY7VqXUG7hipgdWaU
                  m2nhnzdvxJf5KJo9vjP2nABX65c5sFsWsV8oXcbpehtJi
public version  : xpub661MyMwAqRbcFpYYiuvZpKjKhnJD-
ZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                  DGcpfT56AMFeo8M8KPkFMfLUtvwjwb6WPv8rY65L2q8Hz
tree depth      : 0
fingerprint     : 9d9c6092
```

```
parent f'print  : 00000000
child index     : 0
chain code      :
80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
private key     : yes
secret exponent :
112471538590155650688604752840386134637231974546906847202389294096567806844862
 hex            :
f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
wif             : L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBRWfxAzUWwRiGx1kV4sP
 uncompressed   : 5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5Nmb
public pair x   :
764606382405464783648433974782784681018771177678734621270215603682901140160 34
public pair y   :
598078796574697741020401202982722077309212917366332477370774067536768257777 01
 x as hex       :
a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 y as hex       :
843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
y parity        : odd
key pair as sec :
03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 uncompressed   :
04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322

843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
hash160         : 9d9c609247174ae323acfc96c852753fe3c8819d
 uncompressed   : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
 uncompressed   : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

Создание ключа по стандарту BIP-32 из парольной фразы:

В этом примере используется очень простая парольная фраза, которую легко угадать.

```
$ ku P:foo

input           : P:foo
network         : Bitcoin
wallet key      :
xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
                    ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
public version  : xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWo-
QayvtS
                    VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
tree depth      : 0
fingerprint     : 5d353a2e
parent f'print  : 00000000
child index     : 0
chain code      :
5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc
private key     : yes
```

```
secret exponent :
65825730547097305716057160437970790220123864299761908948746835886007793998275
 hex            :
91880b0e3017ba586b735fe7d04f1790f3c46b818a2151fb2def5f14dd2fd9c3
wif             : L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
 uncompressed   : 5JvNzA5vXDoKYJdw8SwwLHxUxaWvn9mDea6k1vRPCX7KLUVWa7W
public pair x   :
81821982719381104061777349269130419024493616650993589394553404347774393168191
public pair y   :
58994218069605424278320703250689780154785099509277691723126325051200459038290
 x as hex       :
b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 y as hex       :
826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
y parity        : even
key pair as sec :
02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 uncompressed   :
04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f

826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
hash160         : 5d353a2ecdb262477172852d57a3f11de0c19286
 uncompressed   : e5bd3a7e6cb62b4c820e51200fb1c148d79e67da
Bitcoin address : 19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
 uncompressed   : 1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT
```

Получить информацию в формате JSON:

```
$ ku P:foo -P -j

{
   "y_parity": "even",
   "public_pair_y_hex":
"826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
   "private_key": "no",
   "parent_fingerprint": "00000000",
   "tree_depth": "0",
   "network": "Bitcoin",
   "btc_address_uncompressed": "1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT",
   "key_pair_as_sec_uncompressed":
"04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d3010a
ea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
   "public_pair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
   "wallet_key": "xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWo
QayvtSVYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy",
   "chain_code":
"5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
   "child_index": "0",
   "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
   "btc_address": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
   "fingerprint": "5d353a2e",
```

```
   "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
   "input": "P:foo",
   "public_pair_x":
"818219827193811040617773492691304190244936166509935893945534043477743931681 91",
   "public_pair_y":
"589942180696054242783207032506897801547850995092776917231263250512004590382 90",
   "key_pair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```


Открытый ключ по стандарту BIP-32:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS-
VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
```

Генерация подключа (субключа):

```
$ ku -w -s3/2 P:foo
xprv9wTErTSkjVyJa1v4cUTFMFkWMe5eu8ErbQcs9xajn-
sUzCBT7ykHAwdrxvG3g3f6BFk7ms5hHBvmbdutNmyg6iogWKxx6mefEw4M8EroLgKj
```

Усиленный подключ (субключ):

```
$ ku -w -s3/2H P:foo
xprv9wTErTSu5AWGk-
DeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRCJo79dsVctcsCHsnZ3XU3DhsuaGZbDh8iDkBN45k67U
KsJUXM1JfRCdn1
```

Формат WIF:

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

Адрес:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZzbAii
```

Генерация связки подключей (субключей):

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYn-
Sekc8bYuJS8G3bhXxKWB89Ggn2dzLcoJsuEdRK
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7WJZPnYAmvc77gUQ-
Vej5QHckc5Adtwxa28ACmANi9XhCrRvtFqQcUxt8rUgFz3souMiDdWxJDZnQxzx
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3PTxQFkZJAi-
woUpmvtrxKZu4zfsnr3pqyy2vthpkwuoVq
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuq-
xYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2jtFXdiEY8UsR
NJfqK6DAdyZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDfyBXmZw4jEYXU-
HYc9fT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XFHKkSsaYKWKJbR54bnyAD9GzjUY-
bAYTtN4ruo
```

Генерация соответствующих адресов:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdHnPUdLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVEM
```

Генерация соответствующих форматов WIF:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lvwg14cLJW7QwWPzCtKHdWMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFSgwCyuuvFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Проверка правильности работы с помощью выбора строки по стандарту BIP-32 (соответствующей подключу 0/3):

```
$ ku -W xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuq-
xYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
$ ku -a xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuq-
xYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
```

Проверка прошла, все работает правильно.

Генерация по секретной экспоненте:

```
$ ku 1

input           : 1
network         : Bitcoin
secret exponent : 1
 hex            : 1
wif             : KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHnoWn
 uncompressed   : 5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x   :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y   :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex       :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex       :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity        : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed   :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
```

```
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160           : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed     : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin address   : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
 uncompressed     : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

Версия Litecoin:

```
$ ku -nL 1

input             : 1
network           : Litecoin
secret exponent   : 1
 hex              : 1
wif               : T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
 uncompressed     : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVTwwN4mxLBFrao69XQ
public pair x     :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y     :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex         :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex         :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity          : even
key pair as sec   :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed     :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160           : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed     : 91b24bf9f5288532960ac687abb035127b1d28a5
Litecoin address  : LVuDpNCSSj6pQ7t9Pv6d6sUkLKoqDEVUnJ
 uncompressed     : LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubdM
```

Формат WIF Dogecoin:

```
$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qrioRbQmjxac5TvoTtZuot
```

Из открытой пары (в сети Testnet):

```
$ ku -nT
55066263022277343669578718895168534326250603453777594175500187360389116729240,even

input                  :
55066263022277343669578718895168534326250603453777594175500187360
                           89116729240,even
network                : Bitcoin testnet
public pair x          :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y          :
32670510020758816978083085130507043184471273380659243275938904335757337482424
```

```
 x as hex                :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex                :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity                 : even
key pair as sec          :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed            :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160                  : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed            : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin testnet address  : mrCDrCybB6J1vRfbwM5hemdJz73FwDBC8r
 uncompressed            : mtoKs9V381UAhUia3d7Vb9GNak8Qvmcsme
```

Из HASH160:

```
$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Bitcoin
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
```

В виде адреса Dogecoin:

```
$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6

input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Dogecoin
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE
```

Утилита для работы с транзакциями (tx)

Утилита командной строки tx выводит содержимое транзакций в форме, удобной для чтения человеком, извлекая базу транзакций из кэша транзакций библиотеки pycoin или получая ее от веб-сервисов (в настоящее время поддерживаются blockchain.info и biteasy.com), объединяет транзакции, добавляет или удаляет фрагменты входных и выходных данных, а также позволяет подписывать транзакции.

Ниже приводятся примеры практического применения утилиты tx.

Просмотр общеизвестной «pizza»-транзакции:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
warning: consider setting environment variable PYCOIN_CACHE_DIR=~/.pycoin_cache
to cache transactions fetched via web services
warning: no service providers found for get_tx; consider setting environment
variable PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCK-EXPLORER
usage: tx [-h] [-t TRANSACTION_VERSION] [-l LOCK_TIME] [-n NETWORK] [-a]
          [-i address] [-f path-to-private-keys] [-g GPG_ARGUMENT]
          [--remove-tx-in tx_in_index_to_delete]
```

```
        [--remove-tx-out tx_out_index_to_delete] [-F transaction-fee] [-u]
        [-b BITCOIND_URL] [-o path-to-output-file]
        argument [argument ...]
tx: error: can't find Tx with id
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
```

Оказывается, у нас не настроены веб-сервисы. Выполним настройку:

```
$ PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
$ export PYCOIN_CACHE_DIR PYCOIN_SERVICE_PROVIDERS
```

Настройка не выполняется автоматически, чтобы через утилиту командной строки не произошла утечка секретной информации об интересующих пользователя транзакциях на сайт третьей стороны. Если вас это не беспокоит, то можете поместить приведенные выше три строки в свой *.profile*.

Попробуем выполнить команду еще раз:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Version:  1  tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a  159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
  0:                          (unknown) from
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
Output:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total output 10000000.00000 mBTC
including unspents in hex dump since transaction not fully signed
010000000141045e0ab2b0b82cde-
faf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a493046022100a7f26eda8749
31999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e9199238eb73f07c8f2
09504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e80000001976a9147e
c1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

** can't validate transaction as source transactions missing
```

Последняя строка появляется после процедуры валидации подписей транзакции, так как необходимо указать источник транзакций. Поэтому добавим в команду ключ -a для добавления к транзакциям источника информации:

```
$ tx -a 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
warning: transaction fees recommendations casually calculated and estimates may be incorrect
warning: transaction fee lower than (casually calculated) expected value of 0.1 mBTC,
transaction might not propogate
Version:  1  tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a  159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
  0: 17WFx2GQZUmh6Up2NDNCEDk3deYomdNCfk from
```

```
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
10000000.00000 mBTC  sig ok
Output:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total input  10000000.00000 mBTC
Total output 10000000.00000 mBTC
Total fees        0.00000 mBTC

010000000141045e0ab2b0b82cde-
faf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a493046022100a7f26eda8749
31999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e9199238eb73f07c8f2
09504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e80000001976a9147e
c1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

all incoming transaction values validated
```

Теперь рассмотрим подробнее неизрасходованные выходные данные для заданного адреса (UTXO). В блоке #1 мы видим coinbase-транзакцию, направленную по адресу `12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX`. Воспользуемся утилитой `fetch_unspent`, чтобы найти все биткойны по этому адресу:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/
0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/
31/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/
86/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
a66dddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cefe68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/
5/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/
1/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/100000
fd87f9adebb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/2000000

0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b538e85
3519c726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e0a604f8141
781e62294721166bf621e73a82cbf2342c858eeac/5000000000
```

Приложение Ж

Команды проводника биткойна bx

Проводник биткойна Bitcoin Explorer (bx) – это инструмент командной строки, который предлагает набор разнообразных команд для управления ключами и создания транзакций. Проводник биткойна bx является частью библиотеки libbitcoin.

```
Usage: bx COMMAND [--help]

Info: The bx commands are:

address-decode
address-embed
address-encode
address-validate
base16-decode
base16-encode
base58-decode
base58-encode
base58check-decode
base58check-encode
base64-decode
base64-encode
bitcoin160
bitcoin256
btc-to-satoshi
ec-add
ec-add-secrets
ec-multiply
ec-multiply-secrets
ec-new
ec-to-address
ec-to-public
ec-to-wif
fetch-balance
fetch-header
fetch-height
```

```
fetch-history
fetch-stealth
fetch-tx
fetch-tx-index
hd-new
hd-private
hd-public
hd-to-address
hd-to-ec
hd-to-public
hd-to-wif
help
input-set
input-sign
input-validate
message-sign
message-validate
mnemonic-decode
mnemonic-encode
ripemd160
satoshi-to-btc
script-decode
script-encode
script-to-address
seed
send-tx
send-tx-node
send-tx-p2p
settings
sha160
sha256
sha512
stealth-decode
stealth-encode
stealth-public
stealth-secret
stealth-shared
tx-decode
tx-encode
uri-decode
uri-encode
validate-tx
watch-address
wif-to-ec
wif-to-public
wrap-decode
wrap-encode
```

Более подробную информацию о командах можно найти на домашней странице проводника Bitcoin Explorer (https://github.com/libbitcoin/libbitcoin-explorer) и в пользовательской документации по адресу https://github.com/libbitcoin/libbitcoin-explorer/wiki.

Примеры практического использования команд проводника bx

Рассмотрим несколько примеров использования команд проводника Bitcoin Explorer, в которых поэкспериментируем с ключами и адресами.

Сгенерируем случайное значение источника энтропии с помощью команды seed, которая использует генератор случайных чисел операционной системы. Затем передадим сгенерированное значение источника в команду ec -new для генерации нового секретного ключа. Сохраним поток стандартного вывода в файле private_key:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

Теперь сгенерируем открытый ключ по этому секретному ключу, воспользовавшись командой ec-to-public. Мы передаем файл *private_key* в поток стандартного ввода и сохраняем поток стандартного вывода выполняемой команды в новом файле *public_key*:

```
$ bx ec-to-public < private_key > public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

Мы можем изменить формат public_key, преобразовав его в адрес с помощью команды ec-to-address. Здесь также передается файл *public_key* в поток стандартного ввода:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHyCP8Kw7xQad1Lr6XuzWUnkG
```

Ключи, сгенерированные таким способом, создают недетерминированный кошелек типа type-0. Это означает, что каждый ключ генерируется из независимого источника энтропии seed. Команды Bitcoin Explorer также могут генерировать ключи детерминированного кошелька в соответствии со стандартом BIP-32. В этом случае «главный» ключ (master key) создается из случайного источника энтропии, затем выполняется детерминированное расширение для получения дерева подключей, в результате чего получается детерминированный кошелек типа type-2.

Сначала выполним команды seed и hd-new для генерации главного ключа, который будет использоваться как основа для порождения иереархии ключей:

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < seed > master
$ cat master
```

```
xprv9s21ZrQH143K2BEhMYpNQoUvAgiEjArAVaZaCTgsaGe6LsAnwu-
beiTcDzd23mAoyizm9cApe51gNfLMkBqkYoWWMCRwzfuJk8RwF1SVEpAQ
```

Теперь выполним команду `hd-private` для генерации усиленного ключа «учетной записи» и последовательности из двух секретных ключей в этой учетной записи:

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueu-
fuyQ8Qi3qpjk4aKsbmbfxwcgS8PYbgoR2NWHeLyvg4DhoEE68A1n

$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnBiDbvG4LgndirD-
sia1e9F3DWPkX7Tp1V1u97HKG1FJwUpU

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzbLB4fzHFd6VqCLPGRZFsdjsuMVERadbgDb-
ziCRJru9n6tzEWrASVpEdrZrFidt1RDfn4yA3
```

Далее воспользуемся командой `hd-public` для генерации соответствующей последовательности из двух открытых ключей:

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuk-
tiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz5wza
SW4FiGrseNDR4LKqTy

$ bx hd-public --index 1 < account
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWy-
MaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQDhohEcDFTEYMvYzwRD7Juf8
```

Эти открытые ключи также могут быть сгенерированы из соответствующих им секретных ключей с помощью команды `hd-to-public`:

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuk-
tiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz5wza
SW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWy-
MaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQDhohEcDFTEYMvYzwRD7Juf8
```

Мы можем сгенерировать практически неограниченное количество ключей в детерминированной цепочке, и все они будут порождены из одного источника seed. Такая методика используется во многих приложениях кошельков для генерации ключей, которые могут быть сохранены в резервной копии и восстановлены при помощи единственного значения источника seed. Это проще, чем создавать резервную копию кошелька со всеми случайно сгенерированными ключами каждый раз, когда генерируется новый ключ.

Значение источника seed может быть закодировано (зашифровано) с помощью команды `mnemonic-encode`:

```
$ bx hd-mnemonic < seed > words
adore repeat vision worst especially veil inch woman cast recall dwell appreciate
```

Декодирование (расшифровка) значения seed выполняется командой `mnemonic-decode`:

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

Мнемоническое кодирование упрощает сохранение и даже запоминание значения источника seed.

Предметный указатель

Об авторе

Андреас М. Антонопулос (Andreas M. Antonopoulos) – известный специалист в области ИТ, серийный предприниматель, ставший одной из наиболее знаменитых и уважаемых личностей в области биткойн-технологии. Талантливый оратор, преподаватель, писатель, Андреас помогает понять сложные темы и делает их доступными для всех. В качестве консультанта он помогает технологическим стартапам определять, оценивать и регулировать потенциальные риски для безопасности и бизнеса.

Андреас буквально вырос вместе с Интернетом, создав свою первую компанию по сопровождению BBS (распространенный в то время сервис Bulletin Board System) и прототипа провайдерского сервиса, будучи еще подростком у себя на родине в Греции. Он получил степень по специальности «Информационные технологии, передача данных и распределенные системы» в лондонском University College London (UCL), в последнее время входящем в 10 самых престижных университетов мира. После переезда в США Андреас стал сооснователем и одним из руководителей успешной компании, занимающейся технологическими исследованиями, и в этой роли консультировал десятки компаний из списка Fortune 500 по вопросам организации сетей, обеспечения безопасности, сопровождения центров данных и облачных вычислений. Более 200 его статей на темы обеспечения безопасности, организации облачных вычислений и поддержки центров данных опубликованы в печатном виде и распространены по всему миру. Андреас является владельцем двух патентов (организация сетевой среды и обеспечение безопасности).

В 1990 году Андреас начал преподавательскую деятельность по различным ИТ-темам для частных, профессиональных и академических сред. Он совершенствовал свое ораторское мастерство перед аудиториями различных размеров: от пяти слушателей в классной комнате до нескольких тысяч людей на крупных конференциях. Имея за плечами более 400 официальных приглашений для публичных выступлений, Андреас заслужил признание как харизматичный оратор и преподаватель высшей квалификации. В 2014 году он был приглашен в преподавательсткий состав университета Никосии (Кипр), первого университета в мире, присваивающего ученые степени в области цифровых валют. В этой роли Андреас помог разработать учебные программы (курсы) и стал одним из преподавателей курса «Введение в цифровые валюты» (Introduction to Digital Currencies), предлагаемого как общедоступный открытый онлайновый курс (MOOC) никосийским университетом.

Как биткойн-предприниматель Андреас основал несколько предприятий, использующих биткойны, и стал основателем нескольких проектов с открытым исходным кодом. В настоящее время он является консультантом нескольких компаний, связанных с использованием биткойна и прочих криптовалют. Андреас широко известен как часто публикуемый автор статей и сообщений в блогах по теме биткойна, он является постоянным ведущим популярного подкаста Let's Talk Bitcoin и часто выступает как докладчик на конференциях по технологии и безопасности по всему миру.

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Андреас М. Антонопулос

Осваиваем биткойн

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Снастин А. В.*
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. 40,125. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**