

Отзывы о 'Mastering Bitcoin'

"Когда я рассказываю о Bitcoin широкой аудитории, иногда меня спрашивают 'но как это работает на самом деле?'. Теперь у меня есть отличный ответ на этот вопрос, потому что любой, кто читает *Mastering Bitcoin* получит глубокое понимание того, как это работает, и будет хорошо подкован для написания следующего поколения удивительных криптовалютных приложений".

— Gavin Andresen, Chief Scientist Bitcoin Foundation

"Технологии Bitcoin и blockchain превращаются в фундаментальные строительные блоки для следующего поколения интернета. Лучшие и ярчайшие люди Кремниевой Долины работают над ними. Книга Андреаса поможет вам присоединиться к революции программного обеспечения в мире финансов".

— Naval Ravikant, Co-founder AngelList

"*Mastering Bitcoin* — это лучший технический справочник по технологии Bitcoin на сегодняшний день, а Bitcoin в ретроспективе может рассматриваться в качестве наиболее важной технологии этого десятилетия. Таким образом, эта книга должна стать настольной для каждого разработчика, особенно того, который заинтересован в создании приложений с протоколом Bitcoin. Настоятельно рекомендую".

— Balaji S. Srinivasan (@balajis), General Partner; Andreessen Horowitz

"Изобретение Bitcoin Blockchain произвело собой совершенно новую платформу, настолько же широкую и разнообразную, как и сам Интернет. В качестве одного из ведущих мыслителей, Андреас Антонопулос представляется идеальным выбором, в качестве автора для подобной книги".

— Roger Ver, Bitcoin Entrepreneur and Investor

Содержание

Предисловие

Как я решился написать книгу про Биткойн

Впервые я узнал про Биткойн в середине 2011 года. Моей первой реакцией было: "Пфф, деньги для нердов!" и я игнорировал его еще 6 месяцев, не сумев осознать его важности. Такую же реакцию я потом часто наблюдал у многих из самых умных людей с которыми я знаком, и это меня немного утешает. Когда я натолкнулся на Биткойн во второй раз в обсуждении в списке рассылки, я решил прочитать документ с техническим описанием, написанный Сатоси Накамато, чтобы изучить авторитетный источник и понять о чем это все вообще. Я до сих пор помню тот момент, когда я закончил читать те девять страниц, и осознал что Биткойн это не просто цифровая валюта, а сеть доверия, которая может так-же предоставлять основу для гораздо большего. Осознание того что "Биткойн это не деньги, а децентрализованная сеть доверия" запустило меня в четырехмесячное путешествие по интернету, во время которого я поглощал каждый кусочек информации о Биткойне который мог найти. Я стал одержим и очарован, проводя по 12 и более часов в день привязанным к экрану, читая, пища, программируя, и изучая столько сколько мог. Я вышел из этого состояния похудев на 20 килограмм из-за отсутствия правильного питания, твердо решив посвятить себя работе над Биткойном.

Спустя два года, имея за плечами опыт создания небольших стартапов в области Биткойн, я решил, что пришло время написать мою первую книгу. Биткойн стал для меня источником вдохновения, ведь эта технология была самой восхитительной со времен изобретения Интернета. Пришло время поделиться моим знанием с широкой аудиторией.

Предполагаемая аудитория

Эта книга предназначена прежде всего для программистов. Если вы умеете программировать, эта книга расскажет вам как работают криптовалюты, как их использовать и как разрабатывать ПО для работы с ними. Первые несколько глав также могут быть полезными непрограммистам и могут служить введением в Биткойн и криптовалюты.

Условные обозначения, принятые в книге

В этой книге используются следующие условные обозначения:

Курсив

Указывает новые термины, URL-адреса, адреса электронной почты, имена файлов и расширения файлов.

Моноширинный шрифт

Используется для листингов программ, а также внутри параграфов, относящихся к элементам программ, таким как названия переменных или функций, баз данных, типов

данных, переменных окружения, выражений, и ключевых слов.

Моноширинный жирный

Показывает команды или другой текст, который должен быть набран в буквальном смысле пользователем.

Моноширинный курсив

Показывает текст, который должен быть заменен пользовательскими значениями или значениями, определенными контекстом.

TIP

Этот значок означает подсказку, предложение или общие сведения.

WARNING

Этот символ указывает на предупреждение или предостережение.

Примеры кода

Примеры показаны на языке Python, C++, а также с помощью командной строки Unix-подобной ОС, такой как Linux или Mac OS X. Все фрагменты кода доступны в репозитории [GitHub repository](#) в подкаталоге *code* основного репозитория. Форкните код книги, позапускайте примеры кода, или предложите исправления через GitHub.

Все фрагменты кода могут быть воспроизведены на большинстве операционных систем с минимальным набором компиляторов и интерпретаторов для соответствующих языков. Где необходимо, мы обеспечиваем базовые инструкции и пошаговые примеры вывода этих инструкций.

Некоторые фрагменты кода и вывода програм, были переформатированы для печати. Во всех таких случаях линии были разделены символом обратной косой черты (\), за которым следует символ новой строки. Если вы задумаете перепечатать примеры вручную, удалите эти два символа и объедините строчки.

Все фрагменты кода используют реальные значения и расчеты, где это возможно, так что вы сможете их повторить и увидеть те же результаты. Например, приватные ключи и соответствующие публичные ключи и адреса все реальны. Примеры транзакций, блоков, и ссылок на blockchain реально существуют в фактическом blockchain, так что вы можете просмотреть их в любой системе Bitcoin.

Выражение признательности

Эта книга представляет собой усилия и вклад многих людей. Я благодарен за ту помощь, которую я получил от друзей, коллег и даже совершенно незнакомых людей, которые присоединились ко мне в этих усилиях, чтобы написать окончательную техническую книгу о криптовалютах и Bitcoin.

Пожалуй невозможно отделить технологию Биткоин от Биткоин-сообщества, и эта книга об

этой технологии появилась благодаря ее сообществу. Моя работа над этой книгой была вдохновлена, поддержана и вознаграждена сообществом от начала и до конца. Эта книга позволила мне быть частью замечательного сообщества в течение двух лет, и безмерно благодарен за принятие меня этим сообществом. Невозможно упомянуть по имени всех тех людей, которых я встречал на конференциях, мероприятиях, семинарах, митапах, вечеринках и небольших частных собраниях, а также множество тех, кто общался со мной через Twitter, Reddit, на форуме bitcointalk.org и на GitHub, которые оказали влияние на эту книгу. Каждая идея, аналогия, вопрос, ответ и объяснение, которые вы найдете в этой книге, были в какой-то момент вдохновлены, опробованы или улучшены с помощью сообщества. Спасибо всем за вашу поддержку; без вас эта книга не появилась бы на свет. Я бесконечно благодарен.

Путешествие к авторству началось задолго до первой книги, конечно. Мой первый язык (и школьный) был греческий, так что я должен был пройти корректирующий курс английского письменного на первом курсе университета. Я выражаю благодарность Diana Kordas, моему учителю английского письменного, который помог мне обрести уверенность и навыки в том году. Позже, как профессионал, я развил свои навыки технического писателя о дата-центрах, публикуясь в журнале "Network World". Я выражаю благодарность John Dix и Jon Gallant, которые дали мне работу обозревателя в Network World" и моему редактору Michael Cooney и моей коллеге Johna Till Johnson, которые редактировали мои обзоры и делали их пригодными для публикации. Написание 500 слов в неделю в течении четырех лет дали мне достаточно опыта в конечно счете решить стать автором. Спасибо Jean de Vera за ее своевременное поощрение к авторству и за твердую веру и убежденность, что я имею в себе книгу.

Спасибо также тем, кто поддержал меня и предоставил свои рекомендации, когда я предложил мою книгу O'Reilly. В частности, спасибо John Gallant, Gregory Ness, Richard Stiennon, Joel Snyder, Adam B. Levine, Sandra Gittlen, John Dix, Johna Till Johnson, Roger Ver, и Jon Matonis. Особая благодарность Richard Kagan и Tymon Mattoszk, за рецензию ранних версий и Matthew Owain Taylor за финальную обработку.

Благодаря Cricket Liu, автору книги *DNS и BIND*, который познакомил меня с O'Reilly. Спасибо также Michael Loukides и Allyson MacDonald из O'Reilly, который работал в течение нескольких месяцев, чтобы помочь в появлении этой книги. Allyson был особенно терпелив когда сроки были упущены.

Первые несколько набросков первых глав были самыми трудными, потому что сам Bitcoin предмет трудный. Каждый раз, когда я тянул за одну ниточку, на свет появлялся целый клубок. Я неоднократно застревал и чувствовал себя подавленно из-за невозможности объяснить простыми словами столь емкую тему. В конце концов, я решил рассказать историю Bitcoin через истории людей, использующих криптовалюту и книгу стало намного проще писать. Я благодарю своего друга и наставника Ричарда Кэгана, который помог раскрутить эту историю и пройти сложные моменты, и Памелу Морган, которая рассмотрела проверяла ранние версии каждой главы и задавала жесткие вопросы, чтобы сделать их лучше. Кроме того, благодарю разработчиков из группы San Francisco Bitcoin Developers Meetup и Taariq Lewis, соучредителя группы, за оказание помощи в проверке раннего материала.

Во время написания этой книги, я сделал доступными ранние черновики на сервисе GitHub и

предложил их для публичного комментирования. Я получил более ста замечаний, предложений, исправлений, и дополнений. Эти предложения вместе с моими благодарностями доступны в [Черновик раннего доступа](#). Особая благодарность Minh T. Nguyen, который вызвался управлять предложениями в GitHub и сделал значительный вклад от самого себя. Спасибо также Andrew Naugler за дизайн инфографики.

После того, как книга была разработана, она прошла через несколько раундов технического обзора. Благодарю Cricket Liu и Lorne Lantz за тщательного обзор, комментарии и поддержку.

Несколько Bitcoin-разработчиков предложили примеры кода, отзывы, комментарии и поддержку. Благодарю Amir Taaki и Eric Voskuil за примеры фрагментов кода и большое количество замечаний; Vitalik Buterin и Richard Kiss за помощь с математикой эллиптических кривых и код некоторых примеров; Gavin Andresen за исправления, комментарии и поддержку; Michalis Kargakis за комментарии, вклад и описание btcd; и Robin Inge за поиск опечаток и улучшения печатной версии.

Я обязан своей любовью к словам и книгам моей матери, Терезе, которая вырастила меня в доме, где книги стояли вдоль каждой стены. Моя мама также купила мне мой первый компьютер в 1982 году, несмотря на то, что считала себе технофобом. Мой отец, Менелаос, инженер-строитель, который недавно опубликовал свою первую книгу в возрасте 80 лет, был тем, кто научил меня логическому и аналитическому мышлению и любви к науке и технике.

Спасибо всем за поддержку на протяжении всего этого пути.

Черновик раннего доступа

Многие авторы предложили комментарии, исправления и дополнения к проекту раннего выпуска на GitHub. Спасибо всем за ваш вклад в эту книгу. Ниже приведен список известных авторов GitHub, включая их GitHub-идентификаторы в скобках:

- Minh T. Nguyen, GitHub contribution editor (enderminh)
- Ed Eykholt (edeykholt)
- Michalis Kargakis (kargakis)
- Erik Wahlström (erikwam)
- Richard Kiss (richardkiss)
- Eric Winchell (winchell)
- Sergej Kotliar (ziggamon)
- Nagaraj Hubli (nagarajhubli)
- ethers
- Alex Waters (alexwaters)
- Mihail Russu (MihailRussu)
- Ish Ot Jr. (ishotjr)

- James Addison (jayaddison)
- Nekomata (nekomata-3)
- Simon de la Rouviere (simondlr)
- Chapman Shoop (belovachap)
- Holger Schinzel (schinzelh)
- effectsToCause (vericoi)
- Stephan Oeste (Emzy)
- Joe Bauers (joebauers)
- Jason Bisterfeldt (jbisterfeldt)
- Ed Leafe (EdLeafe)

Открытая версия

Это открытое издание "Mastering Bitcoin", изданной для перевода под лицензией [Creative Commons Attribution Share-Alike License \(CC-BY-SA\)](#). Эта лицензия позволяет читать, делиться, копировать, печатать, продавать или повторно использовать эту книгу или части этой книги, если соблюдаются следующие условия:

- Применение той же лицензии (Share-Alike)
- Включены атрибуты

Атрибуции

Mastering Bitcoin by Andreas M. Antonopoulos LLC <https://bitcoinbook.info>

Copyright 2016, Andreas M. Antonopoulos LLC

Перевод

Если вы читаете эту книгу на языке, отличном от английского, она была переведена добровольцами. Следующие люди внесли свой вклад в этот перевод:

- Юрий Филипов (<http://coinside.ru>)
- Name 2
- Name 3

Краткий глоссарий

Этот быстрый глоссарий содержит многие из терминов, используемых в связи с Bitcoin. Эти термины используются на протяжении всей книги, поэтому вы можете заходить размещать здесь закладку.

address

Адрес Bitcoin выглядит 1DSrfjdB2AnWaFNgSbv3MZC2m74996JafV. Он состоит из строки букв и цифр, начинающаяся с "1" (число один). Так же, как вы просите кого-то отправить письмо на ваш адрес электронной почты, вы можете попросить других отправить биткоины на ваш Bitcoin адрес.

bip

Предложения по улучшению Bitcoin. Набор предложений по улучшению Биткоин, поданных членами сообщества. Например, BIP0021 представляет собой предложение по улучшению URI-схемы Биткоин.

bitcoin

Название денежной единицы (монеты), сети и программного обеспечения.

block

Группировка транзакций, отмеченная временной меткой, и маркером предыдущего блока. Заголовок блока хешируется для получения доказательства работы, тем самым валидируя транзакции. Валидные блоки добавляются к основному блокчейну на основе сетевого консенсуса.

blockchain

Список проверенных блоков, каждый из которых указывает на своего предшественника вплоть до блока генезиса.

confirmations

После того, как транзакция включается в блок, она получает одно подтверждение. Как только находится *другой* блок по цепочке, транзакция получает уже два подтверждения, и так далее. Шесть или больше подтверждений считаются достаточным доказательством того, что сделка не может быть отменена.

difficulty

Общее для всей сети свойство, управляющее количеством вычислений, которое требуется произвести для доказательства работы.

difficulty target

Сложность, при которой все вычисления в сети будут находить блоки примерно каждые 10 минут.

difficulty retargeting

Общесетевой перерасчет сложности, который происходит каждые 2106 блоков.

fees

Отправитель сделки часто включает комиссионные для оплаты обработки транзакции. Большинство сделок требуют минимальную плату в размере 0.5 mBTC.

hash

Цифровой отпечаток некоторого двоичного входа.

genesis block

Первый блок в blockchain, инициализирующий криптовалюту.

miner

Сетевой узел, который находит доказательство работы для новых блоков путем многократного хэширования.

network

Пиринговая сеть, которая распространяет транзакции и блоки каждому Bitcoin-узлу сети.

Proof-Of-Work

Некоторое количество данных, для нахождения которых требуется совершить значительное количество вычислений. В Bitcoin, майнеры должны найти численное решение алгоритма SHA256, которое бы соответствовало общесетевой целевой сложности.

reward

Количество биткоинов, возникающее с каждым новым блоком в качестве вознаграждения для майнера, который нашел решение для Proof-Of-Work. В настоящее время составляет 25BTC за блок.

secret key (aka private key)

Секретное число, которое отпирает биткоины, посланные на соответствующий адрес. Секретный ключ выглядит как 5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh.

transaction

Проще говоря, передача биткоинов с одного адреса на другой. Более точно, транзакция — это подписанная структура данных, выражающая передаваемое значение. Транзакции передаются по сети Bitcoin, собираются майнерами, и включаются в блоки, составляющие цепочку блоков blockchain.

wallet

Программное обеспечение, которое содержит все ваши Bitcoin-адреса и приватные ключи. С его помощью можно отправлять, получать и хранить биткоины.

Введение

Что такое Биткоин?

Биткоин — это набор концептов и технологий, которые совместно образуют основу для экосистемы цифровых денег. Денежные единицы, называемые биткоинами, используются для сохранения и передачи стоимости между участниками сети. Пользователи Биткоин коммуницируют друг с другом посредством протокола Биткоин через сеть Интернет, однако, протокол может использоваться и внутри какой-либо другой сети. Стек протоколов Биткоин доступен в виде ПО с открытыми исходными текстами и может быть запущен на различных устройствах, включая ноутбуки и смартфоны, что делает технологию легко доступной.

Пользователи имеют возможность перемещать биткоины по сети так же, как это происходит с обычными валютами: покупать и продавать товары, посылать деньги людям или организациям. Биткоины могут быть куплены, проданы либо обменяны на другие валюты на специализированных биржах. В каком-то смысле, Биткоин является идеальной денежной технологией для интернета, так как он быстр, безопасен и не имеет границ.

В отличие от традиционных валют, биткоины полностью виртуальны. Не существует физических монет или даже монет в цифровом представлении. Монеты подразумеваются при передаче ценности от отправителя к получателю. Пользователи Биткоин владеют ключами, которые позволяют доказать права владения в транзакциях Биткоин-сети, разрешая потратить средства и переместить их новому владельцу. Эти ключи часто хранятся в цифровых кошельках на компьютерах каждого пользователя. Владение ключами является единственным необходимым требованием для создания транзакций, а следовательно весь контроль над средствами перекладывается на плечи самих пользователей.

Биткоин — это распределенная одноранговая (P2P) сеть в том смысле, что в ней не существует "центрального" сервера или контрольного узла. Биткоины возникают в процессе "майнинга": конкурентного решения математической задачи, сопутствующего проведению транзакций. Любой участник сети Биткоин (т.е. кто угодно, с запущенным полным стеком протокола) может быть майнером, т.е. использовать вычислительные мощности своего компьютера для подтверждения транзакций. В среднем каждые 10 минут, кто-нибудь подтверждает транзакции за предыдущие 10 минут и вознаграждается за это новосозданными биткоинами. По существу, "майнинг" децентрализует эмитирование валюты и клиринговые функции центрального банка, таким образом заменяя его.

Протокол Bitcoin включает в себя встроенные алгоритмы, которые регулируют функцию майнинга в сети. Сложность вычислительной задачи, решение которой ищут майнеры для того, чтобы успешно записать блок транзакций, регулируется динамически, так что, в среднем, кто-нибудь находит решение каждые 10 минут независимо от того, сколько майнеров (и центральных процессоров) работает над решением задачи в любой момент. Протокол также вдвое уменьшает число биткоинов, возникающий в процессе майнинга каждые четыре года, и ограничивает общее количество биткоинов, которые когда-либо будут созданы числом 21 млн.

монет. В результате этого, количество биткоинов в обращении близко напоминает легко предсказываемую кривую, которая достигает 21 миллионов в 2140 году. Так как скорость эмитирования Биткоин уменьшается, в долгосрочной перспективе, Bitcoin является дефляционной валютой. Кроме того, Bitcoin не может быть инфлирован за счет "печатания" новых денег сверх ожидаемой нормы выдачи.

За кулисами, Bitcoin — это также название протокола и распределенной сети. Биткоин, как валюта — это всего-лишь первое применение данного изобретения. Как разработчик, я вижу Bitcoin сродни Интернету, только для денег, как сеть распространения ценности и обеспечения прав собственности цифровых активов через сеть распределенных вычислений. Потенциал изобретения Bitcoin намного больше, чем кажется на первый взгляд.

В этой главе мы начнем с объяснения некоторых основных понятий и терминов, получения необходимого программного обеспечения и использования Bitcoin для простых операций. В следующих главах мы начнем раскрывать технологию слой за слоем и изучим внутреннюю работу сети и протокола Bitcoin.

Цифровые валюты до Bitcoin

Появление жизнеспособных виртуальных цифровых денег тесно связано с развитием криптографии. Что неудивительно, учитывая фундаментальные проблемы, связанные с использованием битов для представления "ценности", которую можно обменять на товары или услуги. Два базовых вопроса для любого, кто собрался принимать цифровые деньги:

1. Могу ли я доверять тому, что деньги являются подлинными, а не поддельными?
2. Могу ли я быть уверенным, что никто другой не сможет претендовать на то, что эти конкретные деньги принадлежат ему, а не мне? (Так называемая "double-spend" проблема.)

Эмитенты бумажных денег постоянно сражаются с проблемой подделки, используя всё более и более сложную бумагу и технологию печати. Материальные деньги решают "double-spend проблему" легко, так как одна и та же купюра не может находиться в двух местах одновременно. Конечно, конвенциональные деньги зачастую хранятся и передаются в электронном виде. В таких случаях проблемы подделки и двойной траты решаются путём проведения всех электронных транзакций через центральный орган власти, обладающий глобальной картиной всей валюты в обороте. Для цифровых денег, которые не могут воспользоваться преимуществами секретных чернил или голографических полосок, криптография предоставляет базис для доверия легитимности пользовательских претензий на валюту. Конкретно, криптографические цифровые подписи позволяют пользователю подписать цифровое имущество или транзакцию, подтверждая своё владение данным активом. При условии соответствующей архитектуры, цифровые подписи также могут быть использованы для решения проблемы двойной траты.

Когда в конце 1980-х годов криптография начала становиться более широко доступной и понятной, многие исследователи начали пытаться использовать криптографию для создания цифровых валют. Эти ранние проекты цифровых валют издавали цифровые деньги, как правило подкреплённые национальной валютой или драгоценными металлами вроде золота.

Не смотря на то, что ранние цифровые валюты работали, они были централизованными и, как результат, могли быть легко атакованными правительствами или хакерами. Ранние цифровые валюты использовали аналоги централизованных банковских клиринговых систем для разрешения балансов транзакций. К сожалению, в большинстве случаев эти возникшие цифровые валюты беспокоили правительства и вскоре исчезли со сцены. Некоторые из них прекратили существования в результате банкротств их родительских компаний. Для того, чтобы обрести надежность от противников, будь то правительства или криминальные элементы, децентрализованная цифровая валюта должна была избегать единой точки атаки. Биткоин как раз и является подобной системой, с полностью децентрализованным дизайном, без какого-либо центрального пункта управления, который мог бы быть атакован или поврежден.

Биткоин представляет собой кульминацию десятилетий исследований в области криптографии и распределенных систем и включает в себя соединение четырех ключевых инноваций:

- Децентрализованную peer-to-peer сеть (протокол Биткоин)
- Публичную бухгалтерскую книгу транзакций (т.н. блокчейн)
- Децентрализованную математически и детерменистически эмиссию денег (т.н. майнинг)
- Систему децентрализованной верификации транзакций

История Биткоин

Bitcoin был изобретен в 2008 году с публикацией статьи под названием "Bitcoin: Система Peer-to-Peer Электронных Денег," от автора под псевдонимом Сатоши Накамото. Накамото совместил несколько предыдущих изобретений, таких как b-money и HashCash и на этой основе создал полностью децентрализованную систему электронной наличности, которая была бы не зависима от центрального управления для эмиссии средств и проверки сделок. Ключевой инновацией было использовать распределенную систему вычислений (так называемый алгоритм "доказательство работы") для проведения глобальных "выборов" каждые 10 минут, что позволило децентрализованной сети прибывать в состоянии консенсуса о транзакциях. Это элегантно решало вопрос двойной траты, когда одна денежная единица могла бы быть потрачена дважды. Ранее проблема двойной траты была слабым местом цифровых валют, что решалось при помощи классического централизованного клиринга.

Сама сеть Биткоин начала свое существование в 2009-ом году на основе первой реализации

Сатоши Накамото. Впоследствии код был не раз пересмотрен и переписан другими программистами. Сеть распределенных вычислений, которая обеспечивает системе безопасность и надежность, увеличилась в размерах экспоненциально и в данный момент по своей мощности превышает возможности крупнейших суперкомпьютеров. Общая рыночная стоимость Биткоин примерно составляет объем между 5 и 10 миллиардами долларов США в зависимости от текущего курса. Крупнейшей транзакцией, проведенной через сеть на сегодняшний момент явилась таковая объемом 150 миллионов долларов. При этом перевод был осуществлен почти мгновенно и почти ничего его владельцу не стоил.

Сатоши Накамото пропал из публичной видимости в апреле 2011 года, переложив ответственность за разработку системы на плечи группы добровольцев. Кто стоял за личностью виртуального создателя Биткоин, до сих пор неизвестно. Тем не менее, ни Сатоши Накамото, ни кто-либо еще не осуществляет контроль над системой Биткоин, которая работает на основе прозрачных математических принципов. Изобретение само по себе явилось инновационным и уже продвинуло науку в области распределенных вычислений, экономики, и эконометрики.

1. Решение распределенной вычислительной задачи

Изобретение Сатоси Накамото является практическим решением одной ранее нерешенной проблемы распределенных вычислений, известной как "проблема византийских генералов." Вкратце, проблема состоит в попытке договориться о ходе действий путем обмена информацией по ненадежным и даже потенциально скомпрометированным каналам связи. Решение Сатоси Накамото, использует концепцию "доказательства работы" для достижения консенсуса без центрального доверенного центра и представляет собой прорыв в науке о распределенных вычислениях, что имеет широкое применение за пределами криптовалют. Этот принцип может быть использован для достижения консенсуса децентрализованных сетей, для доказательства справедливости выборов, лотерей, реестров активов, цифровому нотариальному заверению и т.д.

Использование Биткоин, пользователи и их истории

Bitcoin — это технология, но она выражает собой деньги, которые по сути являются языком переноса стоимости между людьми. Давайте посмотрим на людей, которые используют Bitcoin и некоторые из самых распространенных видов валюты и протокола через призму их историй. Мы неоднократно будем использовать эти примеры на протяжении всей книги для того, чтобы проиллюстрировать реальные сценарии использования электронных денег и механизмы в их основе.

North American low-value retail

Алиса живет в Области Залива в Северной Калифорнии. Она слышала о Bitcoin от своих друзей технарей и хочет начать его использование. Мы будем следить за ее историей, о том,

как она узнает о Bitcoin, приобретает их, а затем тратит для покупки чашки кофе в кафе Боба в Пало-Альто. Эта история познакомит нас с программным обеспечением, биржами и основные операции с точки зрения конечного потребителя.

North American high-value retail

Кэрол владелица галереи искусств, расположенной в Сан-Франциско. Она продает дорогие картины за Биткойн. Эта история поведует об атаке на консенсус под названием "51%" при торговле предметами с высокой стоимостью.

Offshore contract services

Боб, владелец кафе в Пало-Альто, создает новый сайт. Он заключил контракт с индийским веб-разработчиком по имени Гопеш, который живет в Бангалоре, Индия. Гопеш согласился на оплату в Биткойн. Эта история расскажет об использовании Биткойн для аутсорсинга, контрактных работ и международных денежных переводов.

Charitable donations

Евгения является директором детской благотворительности на Филиппинах. Недавно она открыла для себя Биткойн и хочет использовать его для обращения к группе иностранных и отечественных доноров с целью привлечения дополнительных средств на благотворительность. Она также изучает способы использования Биткойн для распределения средств туда, где это необходимр. Эта история покажет использование Биткойн для глобального трансграничного сбора средств и использования открытой бухгалтерской книги в целях благотворительных организаций.

Import/export

Мухаммед является импортером электроники из Дубая. Он пробует использовать Биткойн для того, чтобы закупать электронику из США и Китая для импорта в ОАЭ, чтобы ускорить процесс платежей. Эта история показывает, как Биткойн может быть использован для больших b2b международных платежей, привязанных к физическим товарам.

Mining for bitcoin

Цзин — студент-компьютерщик из Шанхая. Он построил «ферму» по добыче Биткойн, чтобы подзаработать. Эта история расскажет о "промышленной" основе Биткойн: специализированом оборудовании, используемом для защиты сети и эмиссии новых денег.

Каждая из этих историй основана на реальных людях и реальных примерах использования Биткойн на новых рынках и инновационных решениях глобальных экономических проблем.

Начало работы

Для того, чтобы присоединиться к сети Биткойн и начать использовать валюту, все, что пользователь должен сделать, это загрузить приложение или начать использование веб-приложения. Сам по себе Биткойн является стандартом, но существует много реализаций клиентского программного обеспечения для него. Существует эталонная реализация, также известная как клиент Сатоши, который управляется как проект с открытым исходным кодом

командой разработчиков и является производным от оригинальной реализации, написанной Сатоси Накамото.

Три основные формы Bitcoin клиентов:

Полный клиент

Полный клиент, или "полный узел" — это клиент, который хранит всю историю сделок с Биткоином (каждую сделку по каждому пользователю, когда-либо), управляет кошельком пользователя и может инициировать операции непосредственно над сетью Биткоин. Это похоже на изолированный сервер электронной почты в том смысле, что он охватывает все аспекты протокола, не полагаясь на другие сервера или сторонние сервисы.

Легкий клиент

Легкий клиент хранит кошелек пользователя, но полагается на сторонние сервера для доступа к истории транзакций и самой сети. Легкий клиент не хранит полную копию всех сделок и, следовательно, должен доверять сторонним серверам для проверки транзакций. Это похоже на автономный клиент электронной почты, который подключается к почтовому серверу для доступа к почтовому ящику в том смысле, что он опирается на третью сторону для работы с сетью.

Веб-клиент

веб-клиенты доступны через веб-браузер и хранят кошелек пользователя на сервере, принадлежащем третьей стороне. Это похоже на веб-сервис электронной почты, который полностью полагается на сторонние сервера.

Мобильный Биткоин

Мобильные клиенты для смартфонов, например, на основе ОС Android, могут работать либо в качестве полных клиентов, легких клиентов или веб-клиентов. Некоторые мобильные клиенты синхронизированы с веб- или десктопным клиентом, обеспечивая мультиплатформенный бумажник, установленный на нескольких устройствах, но с общим источником средств.

Выбор Биткоин-клиента зависит от того, как много контроля пользователь хочет над своими средствами. Полный клиент предлагает высочайший уровень контроля и независимости для пользователя, но, в свою очередь возлагает бремя резервных копий и обеспечения безопасности на самого пользователя. На другом конце спектра вариантов находится любой веб-клиент, который является наиболее простым в настройке и использовании, но безопасность и контроль над средствами разделены между пользователем и владельцем веб-сервиса. Если сайт веб-кошелька взломан, а такое бывало часто, пользователи могут потерять все свои средства. И наоборот, если пользователи пользуются полным клиентом не создавая достаточных резервных копий, они могут потерять свои средства вследствие компьютерных аварий.

Для целей этой книги, мы будем продемонстрируем использование различных загружаемых клиентов Биткойн, начиная с эталонной реализации (клиент Сатоси) и заканчивая веб-кошельками. Некоторые из примеров будет требовать использования эталонного клиента, который, в дополнение к тому, что является полным клиентом, также предоставляет API-интерфейсы бумажника, сетевых операций, и транзакций. Если вы планируете исследовать программные интерфейсы в системе Bitcoin, вам понадобится эталонный клиент.

Быстрый старт

Алиса, с которой мы познакомили читателя в [Использование Биткойн, пользователи и их истории](#) не является технически продвинутым пользователем и только недавно услышала о Bitcoin от своего друга. Ее путешествие начинается с посещения официального сайта bitcoin.org, где она находит широкий выбор клиентского ПО. Следуя советам, опубликованным на сайте, она выбирает легкий клиент Multibit.

Алиса переходит по ссылке с bitcoin.org сайте, загружает и устанавливает Multibit. Multibit доступен для компьютеров под управлением Windows, Mac OS и Linux.

WARNING

Биткойн-кошелек должен быть защищен паролем, устойчивым к подбору. Используйте сочетание прописных и строчных букв, цифр и символов. Избегайте использования в паролях личной информации, например даты рождения или названий спортивных команд. Избегайте слов, которые встречаются в словарях любого языка. Вы можете, использовать генератор паролей, чтобы создать совершенно случайный пароль, который бы состоял из не менее 12 символов. Помните, Биткойн — это деньги и они могут быть немедленно перенесены в любую другую точку мира. Если кошелек не защищен, он может быть легко обчищен.

После того, как Алиса скачала и установила приложение Multibit, она запускает его и ее встречает экран приветствия, как показано на [Экран приветствия приложения Multibit](#).

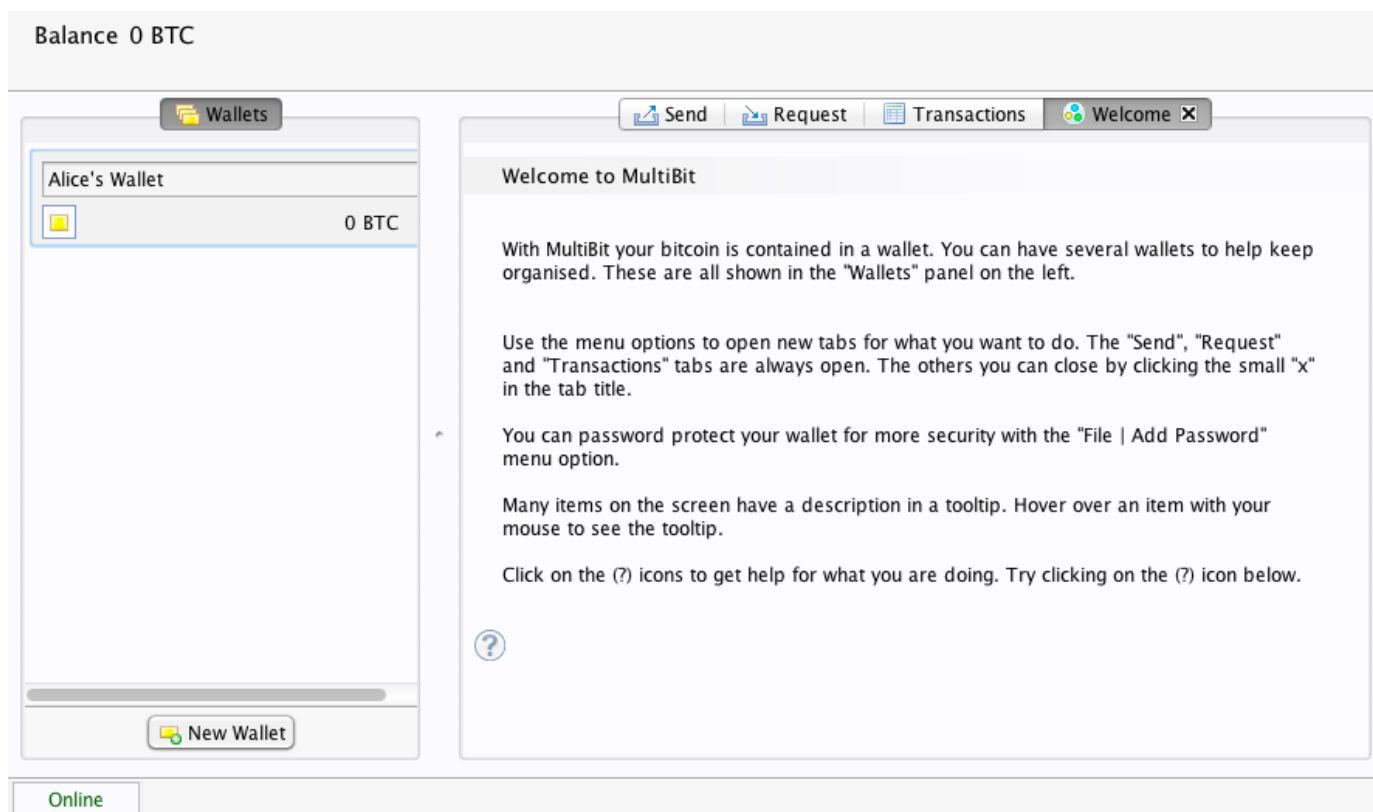


Figure 1. Экран приветствия приложения Multibit

Multibit автоматически создаст кошелек и новый Биткоин-адрес для Алисы, который она сможет увидеть, нажав на вкладку Request, как показано в [Новый Биткоин-адрес Алисы во вкладке Request приложения Multibit](#).

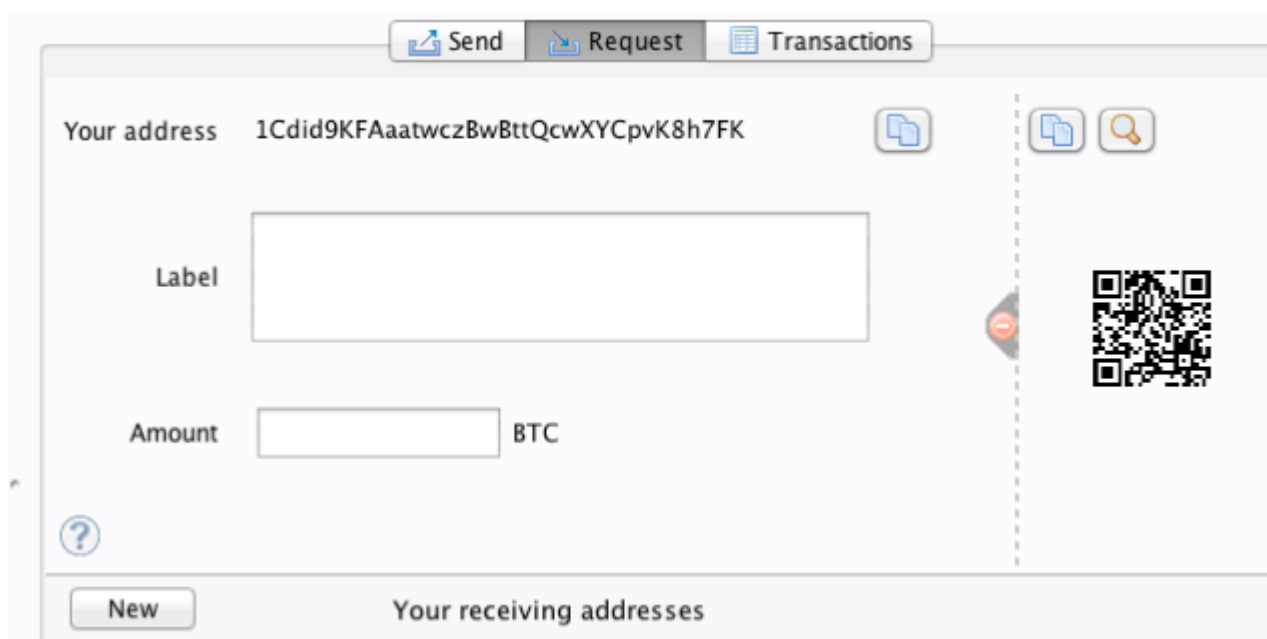


Figure 2. Новый Биткоин-адрес Алисы во вкладке Request приложения Multibit

Наиболее важной частью этого экрана является *Биткоин-адрес* Алисы. По аналогии с адресом электронной почты, Алиса может поделиться этим адресом с кем-либо и любой может

отправить деньги прямо в ее новый кошелек. На экране адрес выглядит как длинная череда букв и цифр вроде: 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Рядом с Биткоин-адресом кошелька отображается его QR-код, который можно сосканировать при помощи камеры смартфона и специального приложения. Нажав на кнопку по соседству с адресом, его можно скопировать в буфер обмена.

Если Алиса распечатает QR-код, скажем, на визитке, то сможет распространять свой Биткоин-адрес при личной встрече и его не потребуется перенабирать.

ТИП

Биткоин-адреса начинаются с цифры 1 или 3. Как и адреса электронной почты, их можно сообщать другим пользователям Биткоин, которые смогут использовать эти Биткоин-адреса для отправки средств непосредственно в ваш кошелек. Но в отличие от адреса электронной почты, вы можете создать сколь угодно новых Биткоин-адресов и все они будут принадлежать вашему кошельку. Кошелек — это просто набор адресов и ключей, которые разблокируют средства. В целях конфиденциальности лучше создавать новый адрес для каждой новой сделки. Ограничений на количество адресов, которые пользователь может создать, практически не существует.

Теперь Алиса полностью готова к использованию своего нового Биткоин-кошелька.

Получаем свои первые биткоины

В данный момент Биткоины не купить в банке или киоске. В 2014, когда писалась эта книга, биткоины было все еще довольно трудно приобрести в большинстве стран. Существует ряд специализированных валютных бирж, где вы можете покупать и продавать биткоины за местную валюту, среди них:

Bitstamp

Европейская валютная биржа, которая поддерживает несколько валют в том числе евро (EUR) и доллары США (USD) посредством банковского перевода.

Coinbase

Американский сайт, предоставляющий услуги веб-кошелька, обменника, а также платформу для продавцов.

Криптовалютные биржи, такие как те, что перечислены выше, работают на пересечении национальных и крипто- валют. Как следствие, они являются предметом национального и международного регулирования. Ваш выбор биржи должен быть обусловлен национальной валютой, которую вы используете, и законодательством вашей страны. Подобно открытию счета в банке, процесс регистрации на криптовалютной бирже может занять несколько дней или даже недель, поскольку потребуется подтверждение идентификации в соответствии с требованием ЗСК (знай своего клиента) и законом по борьбе с отмыванием денег. Если у вас уже есть аккаунт Биткоин-бирже, вы можете покупать и продавать биткоины так же, как если бы это была иностранная валюта с брокерского счета.

Более полный список обменников вы можете найти на сайте [bitcoin charts](#), который собирает котировки и различные другие рыночные показатели с различных бирж.

Существуют четыре других способа получить биткоины для новичков:

- Найти друга, у которого уже есть биткоин и купить некоторое количество у него. Многие пользователи именно так и начинают.
- Использовать доску объявлений наподобие [localbitcoins.com](#) и через нее найти продавца в вашем регионе.
- Продать товар или услугу в обмен на биткоины. Если вы программист, продать свои навыки программирования.
- Использовать Биткоин-банкомат, если таковой есть в вашем городе. Найти Биткоин-банкомат поблизости можно используя онлайн-карту от [CoinDesk](#).

В нашем случае Алису познакомил с Биткоином ее друг, поэтому у нее есть простая возможность получить свои первые биткоины прямо от него, пока ее данные проверяются работниками биржи.

Посылаем и принимаем биткоины

Итак, Алиса создала свой Биткоин-кошелек и теперь готова принимать платежи. Ее кошелек случайным образом сгенерировал приватный ключ (описанный более подробно в [\[private_keys\]](#)) вместе с соответствующим Биткоин-адресом. В данный момент времени, ее Биткоин-адрес не известен Биткоин-сети и нигде не "зарегистрирован". Адрес Биткоин-кошелька — это просто число, которое соответствует ключу, который, в свою очередь, используется для контроля над средствами. Не существует понятия счета, а значит не существует и связи между адресом и счетом. До того момента, пока этот адрес не будет впервые использован в качестве получателя в какой-либо сделке, зарегистрированной в блокчейне, адрес — это просто некое огромное число, одно из огромного количества возможных адресов. Как только адрес оказывается в одной из транзакций, он становится частью известных адресов сети и Алиса получает возможность проверить свой баланс в блокчейне.

Алиса встречается в ресторане со своим другом Джо, который ранее познакомил ее с Биткоином. Во время встречи происходит обмен наличных долларов на биткоины. Алиса взяла с собой распечатку ее адреса и соответствующий ему QR-код. В самом адресе нет ничего важного с точки зрения безопасности. Адрес можно свободно распространять, не рискуя безопасностью кошелька.

Алиса хочет обменять 10 долларов США на биткоины. Она дает Джо \$10 и распечатку своего адреса.

Далее, Джо должен выяснить обменный курс, чтобы перечислить Алисе корректное количество биткоинов. Существует множество приложений и веб-сайтов, отображающих текущий курс. Вот некоторые из самых популярных:

Bitcoin Charts

Сервис, предоставляющий информацию о текущем курсе Биткоин на многих биржах со всего мира, номинированных в разных местных валютах.

Bitcoin Average

А сайт, который показывает простое средневзвешенное курса по каждой из валют.

ZeroBlock

Бесплатное приложение для Android и IOS, отображающее курс Биткоин с различных бирж (см [ZeroBlock, приложение для Android и iOS, отображающее курс Биткоин](#))

Bitcoin Wisdom

Еще один сайт с рыночными данными

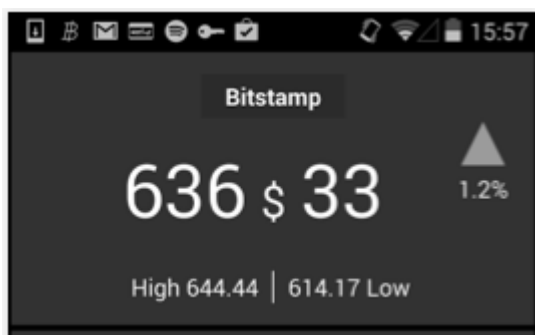


Figure 3. ZeroBlock, приложение для Android и iOS, отображающее курс Биткоин

С помощью одного из приложений или веб-сайтов из нашего списка, Джо определяет, что текущая цена Биткоин составляет примерно 100 долларов. В таком случае, в обмен на \$10 Алисы, Джо должен дать Алисе 0.10 биткоинов, количество, также известное как 100 миллибитов.

После того, как Джо узнал текущий курс, он запускает свое мобильное приложение-кошелек и нажимает кнопку "отправить биткоины". Например, при использовании мобильного кошелька Blockchain для платформы Android, он увидит экран с запросом двух полей, как показано на [Экран мобильного кошелька Blockchain](#).

- Биткоин-адрес назначения
- Количество биткоинов для отправки

В поле ввода адреса Bitcoin, есть маленький значок, который выглядит как QR код. Эта кнопка позволяет Джо сосканировать штрих-код камерой смартфона так, что ему не приходится перенабирать адрес Алисы (1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK), который довольно долго и трудно набрать. Джо нажимает на значок с иконкой QR-кода, после чего сканирует QR-код с распечатки Алисы камерой своего смартфона. Приложение мобильного кошелька заполняет поле Биткоин-адреса и Джо может проверить, что сканирование произошло корректно, путем сравнения нескольких цифр адреса с адресом из распечатки Алисы.

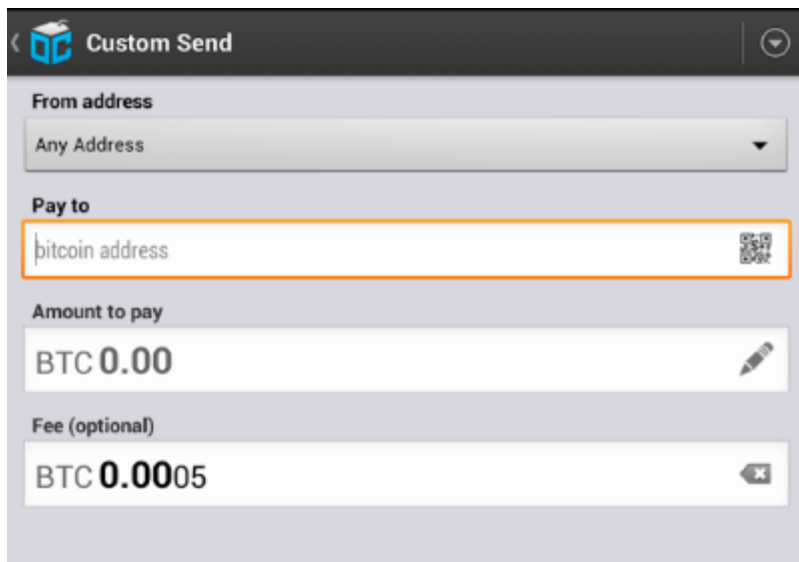


Figure 4. Экран мобильного кошелька Blockchain

Затем Джо вводит 0.10 в поле количества биткоинов для отправки. Он убеждается, что ввел правильное количество, ведь он собирается переслать деньги, а значит любая ошибка может дорого обойтись. Наконец, он нажимает кнопку "Отправить". Мобильный кошелек Джо создает транзакцию, которая снимает 0.10 биткоинов из кошелька Джо и передет их по адресу, указанному Алисой, а затем подписывает сделку приватными ключами Джо. Это сообщает сети Биткоин, что Джо авторизовал передачу ценности с одного из своих адресов на новый адрес Алисы. Далее транзакция передается по протоколу пиринговой сети, и информация об этом быстро распространяется по всей сети Биткоин. Менее чем через секунду, большинство узлов сети уже получило сделку впервые увидело адрес Алисы.

Если у Алисы с собой есть смартфон или ноутбук, она также имеет возможность увидеть транзакцию. Главная бухгалтерская книга Биткоин—это постоянно растущий публично доступный файл, в котором записана каждая сделка, которая когда-либо произошла. Все, что Алиса должна сделать, это поискать в ней свой адрес и убедиться, что на него были отправлены средства. Она может сделать это довольно легко с сайта blockchain.info, просто введя свой адрес в поле поиска. Сайт покажет ей [page](#), на которой перечислены все транзакции с и на этот адрес. Эта страница автоматически обновится как только Джо отправит свои 0.10 биткоинов.

Подтверждения

Сначала адрес Алисы покажет транзакцию от Джо как "неподтвержденную". Это означает, что сделка была распространена по сети, но до сих пор не оказалась в блокчейне, бухгалтерской книге, содержащей список всех транзакций. Для того, чтобы транзакция там оказалась требуется, чтобы ее "подобрал" какой-нибудь майнер и включил в очередной блок. Обычно новый блок возникает примерно каждые 10 минут, оказавшись внутри такого блока, транзакция считается "подтвержденной" и может быть потрачена. Транзакция видна всем мгновенно, но все начинают ей доверять только когда она включена в блок.

Алиса стала гордой владелицей 0.10 биткоинов, которые теперь можно и потратить. В следующей главе мы расскажем о том, как покупать за биткоины и более подробно изучим технологии распространения транзакций.

Как Работает Биткоин

Транзакции, Блоки, Майнинг, и Блокчейн

Система биткоин, в отличие от традиционных банковских и платежных система, построена на децентрализации. Вместо центральных доверительных органов, в биткоине доверие достигается путем взаимодействия различных участников биткоин системы. В этой главе мы посмотрим на биткоин со стороны, проследим путь транзакции в системе биткоин, посмотрим как она становится "доверенной", принимается механизмом распределенного консенсуса и, в конце концов, записывается в блокчейн - распределенный регистр всех транзакций.

Каждый пример основан на реальных транзакциях в биткоин сети, мы будем имитировать взаимодействия между пользователями (Джо, Алиса и Боб) посылая средства с одного кошелька на другой. Мы будем отслеживать транзакции в биткоин сети, используя *blockchain explorer* сайт для отслеживания каждого шага. Блокчейн эксплорер - это веб приложение, работающее как биткоин поисковик, который позволяет найти адреса, транзакции и блоки, чтоб посмотреть детали, статусы и взаимодействия между ними.

Популярные биткоин эксплореры:

- [Blockchain info](#)
- [Bitcoin Block Explorer](#)
- [insight](#)
- [blockr Block Reader](#)

Каждый из них содержит функцию поиска по биткоин адресу, хешу транзакции, и номеру блока. Таким образом пользователь может найти соответствующую информацию из биткоин сети и блокчейна. В каждом примере мы будем приводить ссылку на конкретную запись в биткоин сети, чтоб вы смогли детально её изучить.

Краткий Обзор Биткоина

В диаграмме ниже [Краткий Обзор Биткоина](#) мы видим, что система биткоин состоит из пользователей с кошельками, содержащими ключи, транзакций, которые распространяются по всей сети, и майнеров, которые достигают (путем конкурентных вычислений) консенсуса (всеобщего согласия) в построении блокчейна, который, в свою очередь, является компетентным регистром всех транзакций. В этой главе мы рассмотрим путь одной транзакции в биткоин сети и детально рассмотрим её взаимодействия с каждой частью биткоин системы. В последующих главах мы рассмотрим технологии, стоящие за кошельками, биткоин майнерами и торговыми системами.

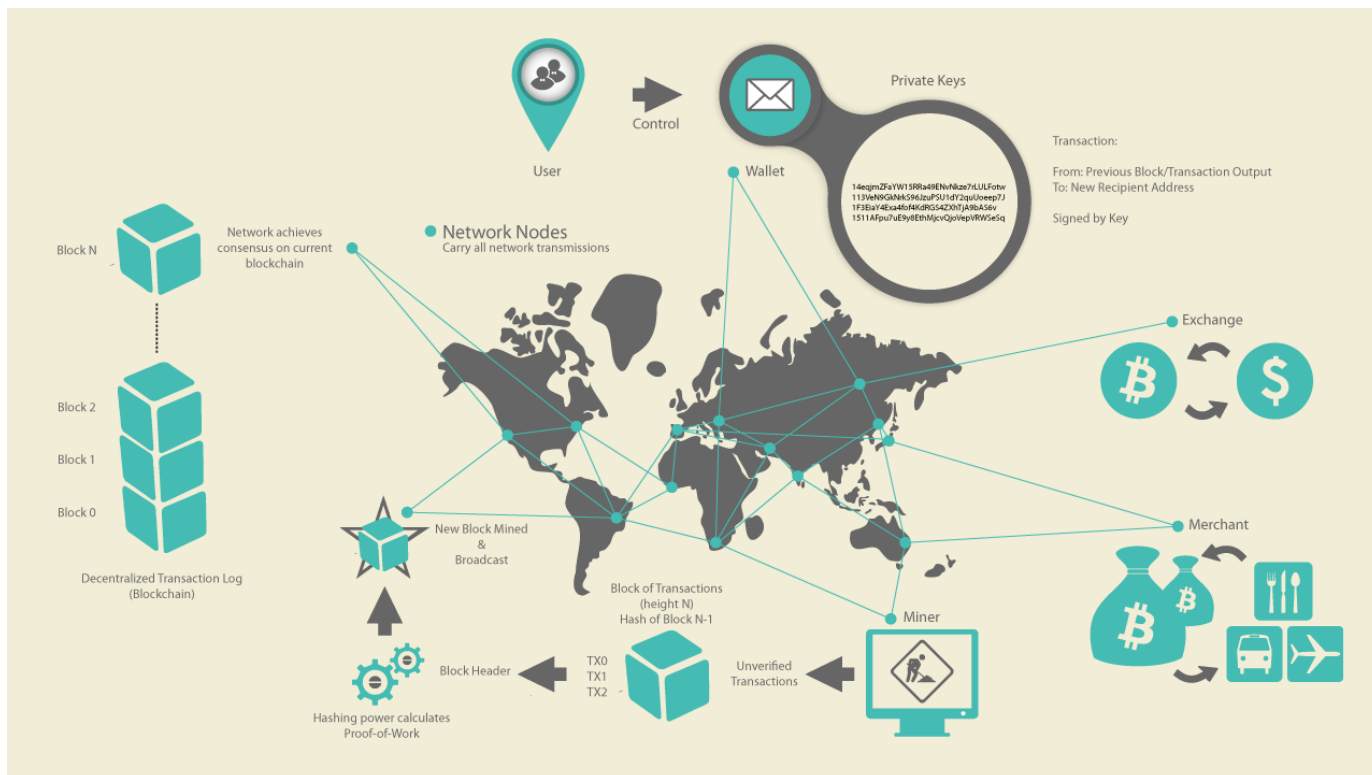


Figure 1. Краткий Обзор Биткоина

Покупая Чашку Кофе

Алиса, с которой вы знакомы из прошлой главы, еще совсем "зеленая", так как недавно получила свой первый биткойн. В [\[getting_first_bitcoin\]](#), Алиса встретила со своим другом Джо, чтоб обменять немного наличных на биткойн. Транзакция, созданная Джо, пополнила кошелек Алисы на 0.10 биткойна. Теперь Алиса отправляется совершить свою первую реальную транзакцию, купив чашку кофе в кафе у Боба в Пало Альто, Калифорния. Боб недавно начал принимать биткойн в своем кафе, добавив опцию оплаты биткойнами в платежный терминал. Цены в кафе указаны в местной валюте (Доллары США), но при оплате клиенты могут воспользоваться опцией оплаты биткойнами. Алиса заказала чашку кофе, Боб ввел заказ в платежный терминал. Терминал отобразил на экране два числа, посчитав стоимость заказа в местной валюте и в биткойнах по текущему курсу, а также показал на экране QR код, который содержит *payment request* для этой транзакции (see [QR код для оплаты \(Подсказка: Отсканируйте\)](#)):

Итого:
\$1.50 USD
0.015 BTC



Figure 2. QR код для оплаты (Подсказка: Отсканируйте)

QR код для оплаты содержит такую ссылку, согласно VIP0021:

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

Компоненты ссылки

Биткоин адрес: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"

Сумма к оплате: "0.015"

Ярлик адреса получателя: "Bob's Cafe"

Описание платежа: "Purchase at Bob's Cafe"

TIP

В отличие от QR кода, который содержит адрес получателя, чек содержит URL платежа, который включает в себя адрес, сумму и описание, как например "Кафе Боба". Это позволяет биткоин кошельку получить необходимую для отправки информацию и отобразить понятную информацию для пользователя. Вы можете отсканировать QR код своим биткоин кошельком, чтоб посмотреть, что увидела Алиса отсканировав такой QR.

Боб говорит, "С вас один доллар пятьдесят, или пятнадцать миллибитов."

Алиса смартфоном сканирует штрихкод с экрана. Её смартфон показывает "0.0150 БТС Кафе Боба" она нажимает кнопку "Отправить" чтоб подтвердить платеж. Примерно за пару секунд (не более того, что заняла бы авторизация пластиковой карты), Боб увидит транзакцию и подтвердит платеж.

В следующих разделах мы рассмотрим эту транзакцию более подробно: увидим как кошелек Алисы ее создал, как она распространялась по сети, как проверялась, и, наконец, как Боб потратил эту сумму в последующих транзакциях.

NOTE

Сеть Биткоин поддерживает переводы дробных количеств, например, от миллибиткоинов (1/1000 часть Биткоин) до 1/100.000.000 доли Биткоин, которая известна как сатоши. В этой книге мы будем использовать термин "биткоины" для обозначения любого количества криптовалюты: от наименьшей единицы (1 сатоши) до максимального возможного числа монет (21.000.000).

Транзакции Биткоин

Простыми словами, транзакция говорит сети, что владелец некоторого количества биткоинов уполномочил передачу определенного количества из них другому владельцу. Теперь новый владелец может потратить эти биткоины путем создания другой транзакции, которая разрешает передачу уже другому владельцу, и так далее, по цепочке смены владельца.

Транзакции аналогичны записям в обычной бухгалтерской книге приходов и расходов. Простыми словами, каждая транзакция содержит один или несколько "входов", из которых поступают средства. С другой стороны транзакции находятся один или больше "выходов", куда средства уходят. Входы и выходы (дебета и кредита) не обязательно в сумме составляют одну и ту же сумму. Выходы содержат чуть меньше, чем сумма на входах за счет "комиссии", небольшой платы майнерам за включения транзакции в блок. Биткоин-транзакция выглядит как две записи в бухгалтерской книге [Транзакция, как двойная запись в бухгалтерской книге](#).

Транзакция также содержит в себе доказательства владения в виде цифровой подписи владельца сумм на каждом из входов. Цифровые подписи владельцев могут быть независимо проверены кем угодно. В терминах Биткоин, "потратить" означает подписание транзакции, перемещающей ценность из какой-либо другой предыдущей транзакции новому владельцу, идентифицируемому Биткоин-адресом.

TIP

Транзакции переносят значения из входов транзакции в выходы транзакции. Значение на входе — это как правило выход какой-то предыдущей транзакции. Выход транзакции назначает нового владельца при помощи обременения, т.е. требования подписи для получения возможности распоряжаться средствами. Выходы из одной транзакции могут быть использованы в качестве входов новых транзакций, тем самым создавая цепочку передачи прав собственности, по мере того, как ценность перемещается от адреса к адресу (см. [Цепочка транзакций, в которой выход одной транзакции является входом для последующей](#)).

Transaction as Double-Entry Bookkeeping							
Inputs		Value		Outputs		Value	
Input 1		0.10 BTC		Output 1		0.10 BTC	
Input 2		0.20 BTC		Output 2		0.20 BTC	
Input 3		0.10 BTC		Output 3		0.20 BTC	
Input 4		0.15 BTC					

Figure 3. Транзакция, как двойная запись в бухгалтерской книге

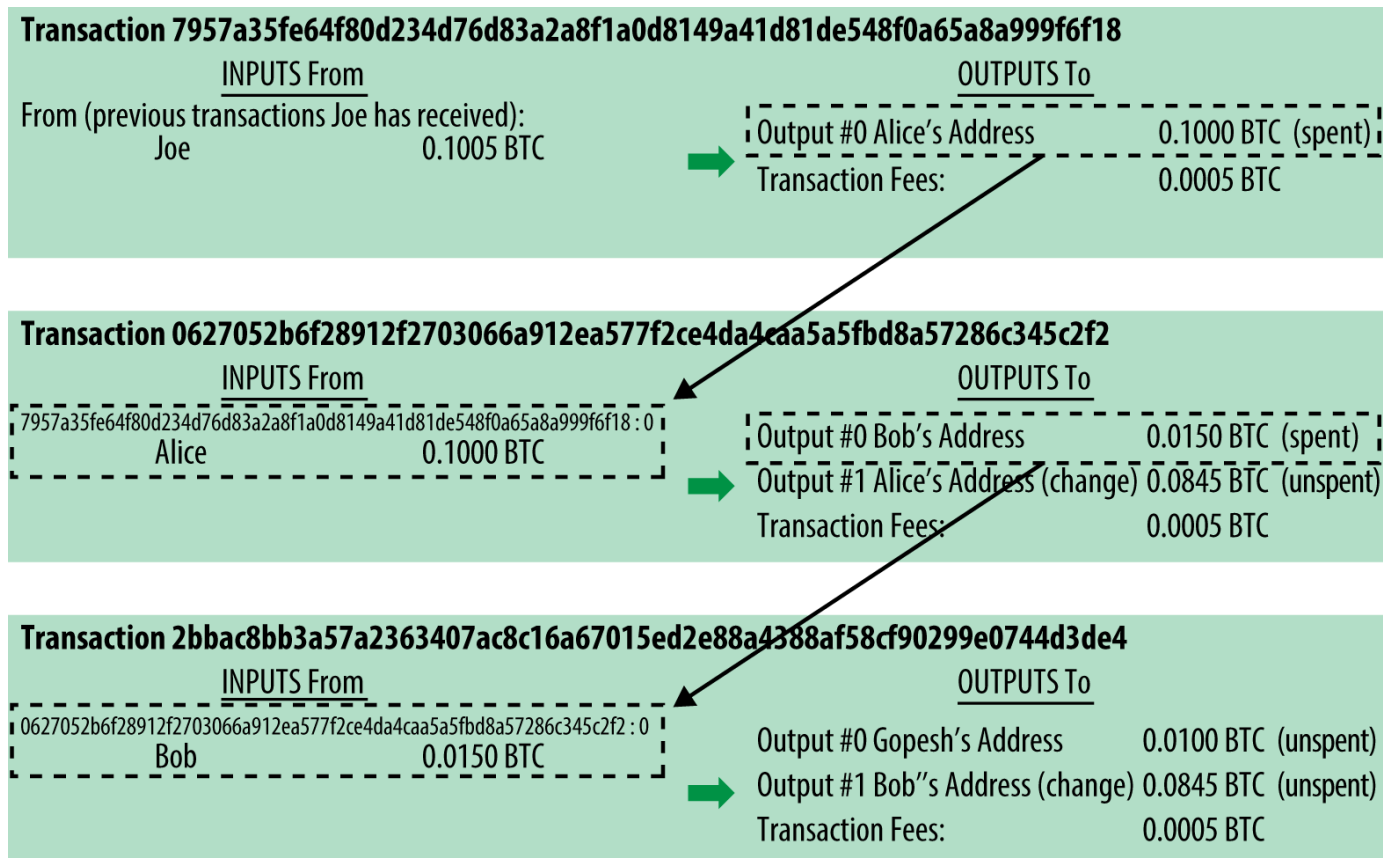


Figure 4. Цепочка транзакций, в которой выход одной транзакции является входом для последующей

Платеж Алисы в кафе Боба использует предыдущую транзакцию в качестве входа. В предыдущей главе Алиса получила биткойны от Джо в обмен на наличные. Та транзакция оказалась заблокирована (обременена) требованием ключа Алисы. Ее новый платеж Бобу ссылается на предыдущую сделку в качестве входа и создает новые выходы для оплаты чашки кофе и для сдачи. Транзакции образуют цепь, в которой входы последующих транзакций соответствуют выходам предыдущих. Ключ Алисы обеспечивает подпись, которая распечатывает эти предыдущие выходы, тем самым доказав остальной сети Биткойн, что Алиса является владелицей средств. Она отправляет оплату за кофе по адресу Боба, добавляя к выходу "обременение" с требованием, что Боб должен предъявить валидную подпись для того, чтобы потратить эту сумму. Это описывает передачу ценности от Алисы к Бобу. Эта цепочка транзакции от Джо к Алисе и далее к Бобу, показана на [Цепочка транзакций, в которой выход одной транзакции является входом для последующей](#).

Обычные виды транзакций

Наиболее распространенной формой транзакции является простой платеж с одного адреса на другой, что часто включает в себя некоторое количество "сдачи" первоначальному владельцу. Этот тип сделки имеет один вход и два выхода как показано на [Самая обычная транзакция](#).

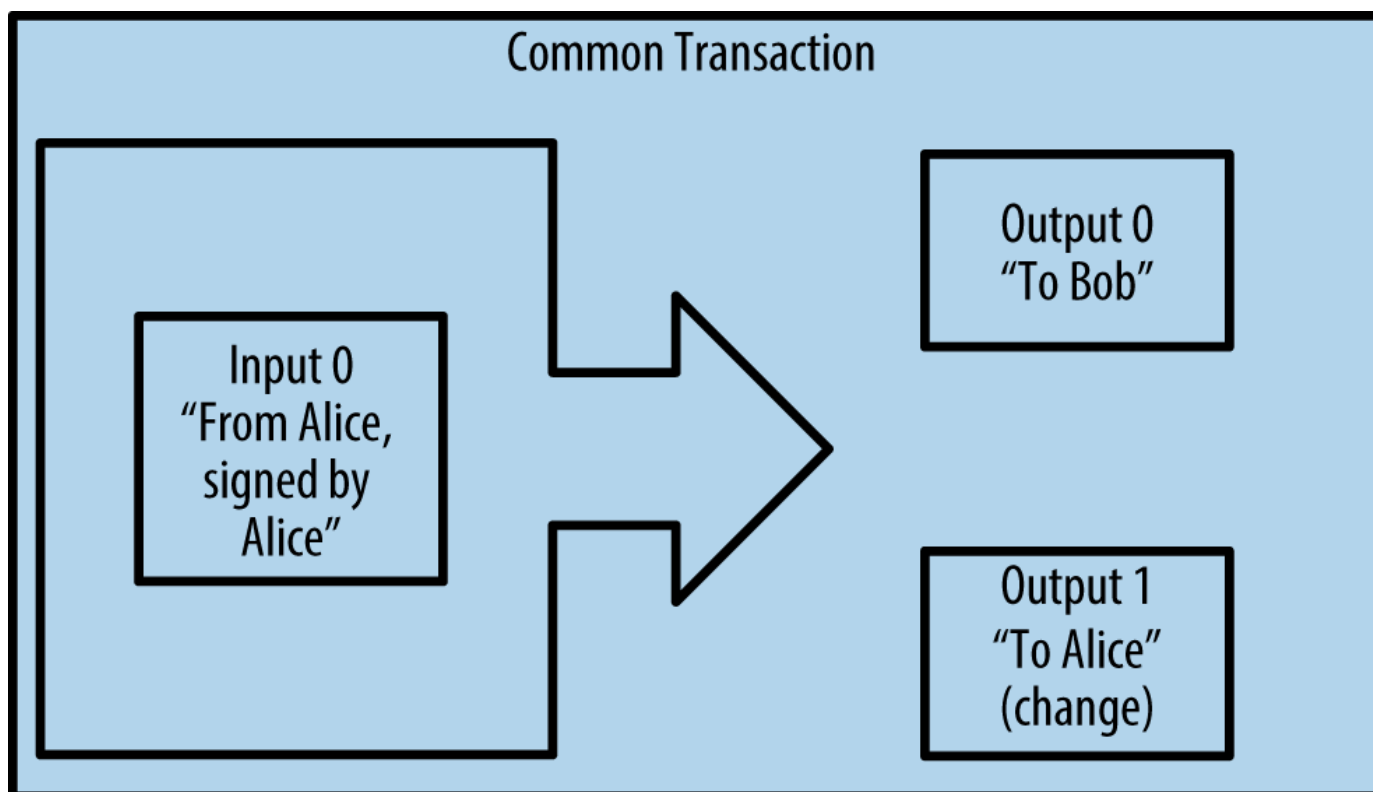


Figure 5. Самая обычная транзакция

Еще один распространенный вид сделки — объединяющий несколько входов в один выход (см. [Транзакция слияния средств](#)). Это представляет собой эквивалент обмена кучи монет на одну купюру в реальной жизни. Подобные транзакции часто создаются ПО кошельков для слияния множества маленьких "сдач".

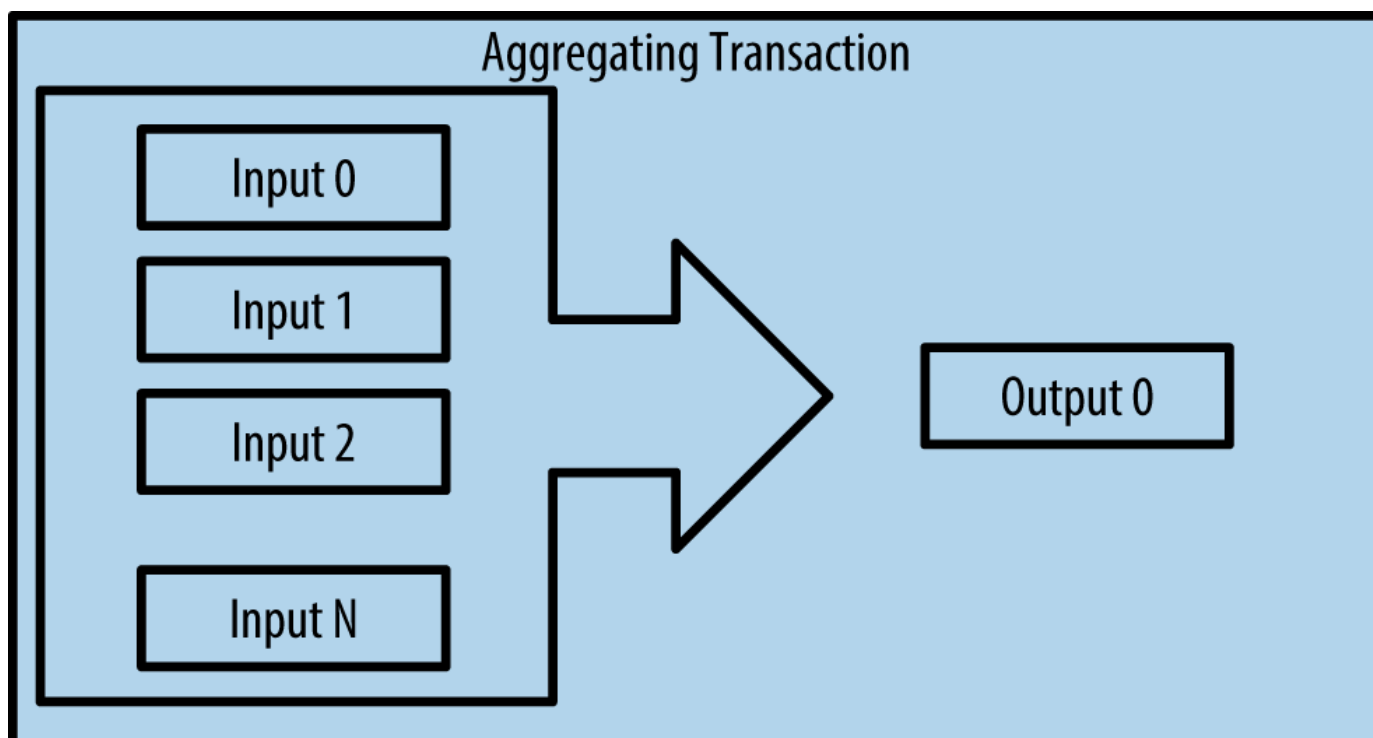


Figure 6. Транзакция слияния средств

Наконец, еще одна распространенная форма транзакция в блокчейне разделяет один вход на несколько выходов, представляющих несколько получателей (см [Транзакция распределяющая средства](#)). Этот тип сделки иногда используется коммерческими структурами для распределения средств на несколько кошельков, например, выплаты зарплат нескольким сотрудникам сразу.

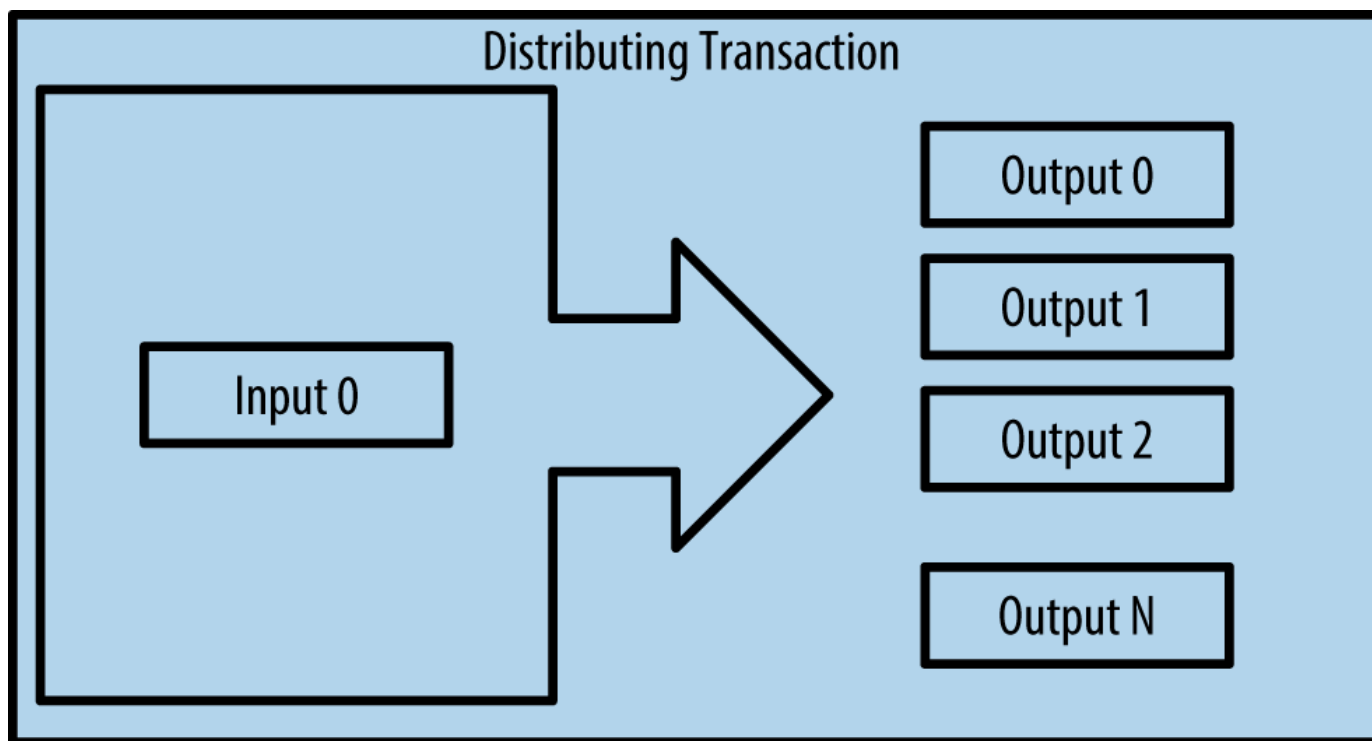


Figure 7. Транзакция распределяющая средства

Создание транзакции

Приложение-кошелек Алисы содержит всю необходимую логику для выбора соответствующих входов и выходов для построения транзакции по заданию Алисы. Алиса должна указать только адрес назначения и количество, а все остальные детали скроет от нее приложение. Важно заметить, что приложение кошелька способно создавать транзакции даже, если оно работает в офлайне. Это примерно так же, как выписать чек находясь дома, а затем отправить его в банк в конверте, т.е. для создания транзакции не обязательно иметь подключение к сети Bitcoin. Однако, для того, чтобы о транзакции узнал весь остальной мир и она вступила в силу подключение все-таки потребуется.

Получение правильных входов

Приложение кошелька Алисы сначала должно найти входы, содержащие достаточное количество средств, чтобы можно было заплатить Бобу. Большинство приложений кошельков содержит небольшую базу данных "неизрасходованных выходов транзакций" ("unspent transaction outputs"), заблокированные (обремененные) собственными ключами кошелька. Значит, кошелек Алисы будет содержать копию выхода транзакции Джо, при помощи которой Алиса приобрела свои первые биткоины (см. [\[getting_first_bitcoin\]](#)). Приложения Биткоин-

кошельков, работающие как клиент с полным индексом на самом деле содержит копию каждого неизрасходованного выхода из каждой транзакции в блокчейне. Это позволяет кошельку не только выбирать входы для транзакций, но также и быстро проверять корректность входов входящих транзакций. Однако, поскольку полный индекс занимает много места на диске, большинство пользователей используют "легкие" кошельки, которые отслеживают только собственные неизрасходованные выходы пользователя.

Если приложение кошелька не содержит копию неизрасходованных выходов транзакций, оно может запросить эту информацию у сети Биткоин через API различных провайдеров, или послав JSON RPC запрос узлу, содержащему полный индекс. [Найти все неизрасходованные выходы по Биткоин-адресу Алисы](#) показывает запрос к RESTful API в виде HTTP GET по определенному URL. Этот URL-адрес вернет все неизрасходованные выходы транзакций адреса, предоставив любому приложению достаточно информации, чтобы можно было построить входы транзакций. Для запроса мы воспользуемся утилитой командной строки *cURL*.

Example 1. Найти все неизрасходованные выходы по Биткоин-адресу Алисы

```
$ curl https://blockchain.info/unspent?active=1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK
```

Example 2. Ответ на запрос

```
{
  "unspent_outputs":[
    {
      "tx_hash":"186f9f998a5...2836dd734d2804fe65fa35779",
      "tx_index":104810202,
      "tx_output_n": 0,
      "script":"76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
      "value": 10000000,
      "value_hex": "00989680",
      "confirmations":0
    }
  ]
}
```

Ответ в [Ответ на запрос](#) показывает один неизрасходованный выход (выход, который еще не был израсходован), на принадлежащем Алисе адресе 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Ответ включает в себя ссылку на транзакцию, в которой найден этот неизрасходованный выход (перевод от Джо) и его значение в 10 млн сатоши, что эквивалентно 0.10 биткоинов. При

помощи этой информации, приложение кошелька Алисы может составить новую транзакцию и отправить средства по новому адресу.

TIP | Просмотр [транзакцию Джо Алисе](#).

Как можно видеть, кошелек Алисы содержит достаточно биткоинов в одном неизрасходованного выходе, чтобы хватило заплатить за чашку кофе. Если бы в одном выходе не хватало средств, то приложению кошелька Алисы, возможно, пришлось бы "рыться" в куче мелких неизрасходованных выходов, как обычно роются в кошельке набирая в ладонь монеты пока не хватит, чтобы заплатить. В обоих случаях может возникнуть необходимость получить некоторое количество сдачи. Об этом мы поговорим в следующем разделе, когда расскажем как приложение кошелька создает выходы транзакции.

Создание выходов

Выходы сделки описываются в виде сценария, который создает обременение на ценность, разрешающее пользование средствами только если сценарий будет исполнен. Проще говоря, выход транзакции Алисы будет содержать сценарий, который говорит что-то вроде, "Этот выход выплачивается тому, кто может представить подпись ключом, соответствующим публичному адресу Боба." Так как только у Боба в кошельке есть ключ, соответствующий этому адресу, только кошелек Боба может представить подобную подписи использовать этот выход. Поэтому Алиса "обременит" выход транзакции требованием предъявления подписи Боба.

Эта транзакция будет также включать в себя второй выход, так как 0.10 BTC слишком много за чашку кофе ценой 0.015 BTC. Алиса должна получить 0.085 BTC в виде сдачи. Выход сдачи Алисы создается *кошельком Алисы* в той же самой транзакции, в которой создается оплата Бобу. По сути, кошелек Алисы делит ее средства на два платежа: один Бобу, и один обратно себе. В одной из более поздних транзакций Алиса сможет потратить остаток сдачи.

Наконец, для того, чтобы транзакция могла быть обработана сетью в разумные сроки, приложение кошелька Алисы присовокупило небольшую комиссионную плату. Комиссия не указана в транзакции явно; подразумевается что ее можно вычислить как разницу между входами и выходами. Вместо 0.085 в качестве сдачи, Алиса создает второй выход для 0.0845, подразумевая в остатке 0,0005 BTC (половина миллибиткоина). Количество 0.10 BTC на входе не полностью потрачено двумя выходами, так как в сумме они дают менее чем 0.10. Результирующая разница является *комиссией*, и уходит в пользу майнера в качестве платы за включение транзакции в блокчейн.

Результирующая транзакция может быть видна через веб-приложение "проводник блокчейна", как показано на [Транзакция Алисы в кафе Боба](#).

Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)



1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA

- (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK -

(Unspent) 0.0845 BTC

97 Confirmations

0.0995 BTC

Summary		Inputs and Outputs	
Size	258 (bytes)	Total Input	0.1 BTC
Received Time	2013-12-27 23:03:05	Total Output	0.0995 BTC
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)	Fees	0.0005 BTC
		Estimated BTC Transacted	0.015 BTC

Figure 8. Транзакция Алисы в кафе Боба

ТИП | Просмотр транзакция Алисы в кафе Боба.

Запись транзакции в бухгалтерскую книгу

Транзакция, которую создал кошелек Алисы имеет размер 258 байт и содержит все необходимое, чтобы подтвердить право собственности на средства и назначить новых владельцев. Теперь транзакция должна быть передана в сеть Биткоин, где она станет частью распределенной бухгалтерской книги (блокчейна). В следующем разделе мы увидим, как транзакция попадает в новый блок и, как блок будет "добыт". Наконец, мы увидим, что по мере добавления новых блоков к блокчейну, доверие к старым возрастает.

Передача транзакции

Так как транзакция содержит в себе всю необходимую информацию, не имеет значения, где и как она будет передана в сеть Биткоин. Сеть Биткоин пиринговая, т.е. каждый Биткоин-клиент одновременно подключен к нескольким другим. Транзакции и блоки в такой топологии быстро достигают всех участников сети.

Как распространяется транзакция

Приложение кошелька Алисы может отправить новую транзакцию любому из других клиентов сети Биткоин в Интернете через любой доступный вид подключения. Ее кошелек не должен подключаться к кошельку Боба напрямую, и она не должна использовать подключение к Интернету, предлагаемое в кафе, хотя оба эти варианта возможны. Любой узел

Биткоин (другой клиент), принявший валидную транзакцию, которую он ранее не видел, обязан немедленно перенаправить ее другим узлам, с которыми он сам держит соединение. Таким образом, транзакция быстро распространяется по всей пиринговой сети и подавляющее число узлов узнает о ней в течение нескольких секунд.

С точки зрения Боба

Если приложения кошелька Боба и Алисы были бы соединены напрямую, то Боб бы получил транзакцию Алисы первым. Однако, даже если кошелек Алисы пошлет транзакцию другим узлам, она достигнет бумажника Боба в течение всего-лишь нескольких секунд. Кошелек Боба немедленно распознает транзакцию Алисы в качестве входящего платежа, так как выходы транзакции будут погашаемы ключами Боба. Кошелек Боба также сможет самостоятельно проверить, что транзакция корректна, использует неистраченные ранее входы, и содержит достаточную сумму комиссионных, чтобы быть включенной в следующий блок. В этот момент Боб может с минимальным для себя риском предположить, что транзакция вскоре будет включен в очередной блок и подтверждена.

ТИП

Распространенное заблуждение о Биткоин-транзакциях состоит в том, что они якобы должны быть "подтверждены" после 10-ти минут ожидания нового блока, или даже 60-ти минут для шести полных подтверждений. Хотя подтверждения и дают гарантию, что транзакция будет принята всей сетью, подобная задержка не является необходимой для платежей малой стоимости, таких как кофе. Торговец может принять валидную транзакцию малой стоимости и без каких-либо подтверждений вообще, с не большим риском, чем при оплате кредитной картой без предъявления удостоверения личности или подписи, как это часто происходит в наши дни.

Добыча Биткоин

Теперь транзакция распространяется по Биткоин-сети. Она не станет записью в общей бухгалтерской книге (*блокчейн*) до тех пор, пока не будет проверена и включена в блок во время процесса, называемого *добычей* (или майнинга). См [\[ch8\]](#) для подробного объяснения.

Система доверия Биткоин основана на вычислениях. Транзакции сохраняются в блоках, которые для своего появления на свет требуют огромное количество вычислений, и лишь небольшое количество вычислений требуется для проверки результатов этих вычислений. Процессе добычи Биткоин служит одновременно двум целям:

- Майнинг создает новые биткоины в каждом новом блоке, почти так же, как центральный банк печатает новые деньги. Количество биткоинов, находимых в новом блоке фиксировано и уменьшается со временем.
- Майнинг создает доверие, гарантируя, что для попадания транзакций в блок требуется достаточно вычислительной мощности. Больше блоков означает больше вычислений, что, в свою очередь, означает больше доверия.

Майнинг можно сравнить с гигантской одновременной игрой в sudoku, но такой, что сбрасывает результат как только кто-то находит решение головоломки и сложность которой автоматически регулируется так, что поиск решения занимает около 10 минут. Представьте себе гигантский sudoku, несколько тысяч строк и столбцов в размере. Если вам показать заполненную цифрами головоломку, то вы сможете достаточно быстро проверить ее корректность. Однако, если в головоломке несколько квадратов заполнено, а остальные пусты, понадобится много работы, чтобы ее решить! Сложность sudoku можно регулировать путем изменения размера (больше или меньше строк и столбцов), но результат решения головоломки довольно легко проверить несмотря на ее размер. "Головоломка", которая используется в Биткоин основана на криптографическом хэше и имеет схожие характеристики: ее асимметрично трудно решить, но легко проверить, а также возможно регулировать ее сложность.

В [\[user-stories\]](#) мы познакомили читателя с Цзин, студентом компьютерного факультета из Шанхая. Цзин участвует в Биткоин-сети в качестве майнера. Каждые 10 минут или около того, Цзин вместе с тысячами других майнеров соревнуется в попытке найти очередной блок. Поиск решения головоломки, так называемое доказательство работы, требует квадриллионы операций хеширования в секунду всей сети Биткоин. Алгоритм доказательства работы представляет собой непрерывающееся хеширование заголовка блока и случайного числа при помощи криптографического алгоритма SHA256 до тех пор, пока не будет найдено решение, удовлетворяющее заранее заданному шаблону. Тот, кто первым из всех майнеров найдет подобное решение, выигрывает этот раунд соревнования и публикует найденный блок в блокчейн.

Цзин начал заниматься майнингом в 2010 году с помощью очень быстрого настольного компьютера. Поскольку все больше шахтеров начали присоединяться к Биткоин-сети, сложность задачи быстро выросла. Вскоре Цзин и другие майнеры перешли на более специализированное железо: игровые видеокарты (GPU). На момент написания этой книги, сложность достигла таких размеров, что заниматься поиском блоков выгодно только на очень специализированном железе, так называемых платах ASIC, в которых используются процессоры созданные для решения лишь одной задачи — поиска хешей. Цзин также присоединился к "майнинговому пулу", который позволяет его участникам разделить между собой усилия по поиску, так и награду за находку. У Цзина в настоящее время работает есть два ASIC в виде двух "компьютеров", подключенных к основному по USB и работающих 24 часа в сутки. Он оплачивает свои расходы за электроэнергию и сверх того немного зарабатывает за счет продажи найденных биткоинов. На его компьютере запущен bitcoind, основной Биткоин-клиент, который требуется для ПО майнинга.

Транзакции майнинга в блоках

Транзакция, переданная по сети не проверяется до тех пор, пока она не станет частью глобальной распределенной бухгалтерской книги, блокчейна. В среднем каждые 10 минут майнеры находят новый блок, который содержит все транзакции с момента последнего блока. От пользователей в сеть постоянно поступают новые транзакции. Как только сеть их получает, они добавляются к временному пулу непроверенных транзакций, поддерживаемому каждым

узлом. Когда майнеры пытаются найти новый блок, они добавляют непроверенные транзакции из этого пула к новому блоку, а затем пытаются найти решение очень трудной задачи (т.е. предъявить доказательство работы). Процесс майнинга будет подробно описан в [\[mining\]](#).

Транзакции будут добавлены в новый блок в порядке приоритета, рассчитанного на основании количества комиссии и нескольких других критериев. Каждый майнер начинает процесс поиска нового блока, как только получает из сети предыдущий блок, понимая что соревнование по поиску предыдущего блока проиграно. Он сразу же создает новый блок, заполняет его транзакциями и идентификатором-отпечатком предыдущего блока, и начинает расчета доказательства работы нового блока. Каждый майнер включает в свой блок специальную транзакцию, в которой вознаграждает самого себя за найденный блок в виде некоторого количества биткоинов (в настоящее время это 25 BTC). Если он находит решение головоломки, что означает, что блок становится валидным, он "выигрывает" это награду, потому что его блок добавляется к глобальному блокчейну, а награждающая транзакция становится расходимой. Цзин, который участвует в майнинговом пуле, настроил свое программное обеспечение так, что награда уходит по адресу пула. Оттуда, доля вознаграждения распределяется Цзин и другим майнерам пропорционально тому объему работы, который каждый вложил в последнем туре.

Сеть подобрала транзакцию Алисы и включила в пул непроверенных сделок. Так как в нее было включено достаточное количество комиссионных, транзакция попала в новый блок, найденный майнинговым пулом Цзин. Примерно через пять минут после того, как транзакция была инициирована кошельком Алисы, ASIC-майнер Цзин нашел решение для блока и опубликовал его в качестве блока #277316, содержащий также 419 других сделок. ASIC-майнер Цзин опубликовал новый блок в сети Биткоин, где другие майнеры подтвердили его и начали новую гонку по поиску следующего блока.

Можно видеть, что блок включает в себя [транзакцию Алисы](#).

Спустя несколько минут, другой майнер нашел новый блок #277317. Так как новый блок основан на предыдущем блоке (#277316), который содержал операцию Алисы, он добавил даже больше вычислений над тем блоком, таким образом усилив доверие к тем транзакциям. Блок, содержащий транзакцию Алисы засчитывается как одно "подтверждение" этой транзакции. Каждый новый блок сверху блока, содержащего сделку является дополнительным подтверждением. По мере того, как блоки нанизываются один на другой, становится экспоненциально сложнее отменить транзакцию, тем самым она становится все более доверительной для сети.

На диаграмме в [Транзакция Алисы, включенная в блок #277316](#) мы видим блок #277316, содержащий транзакцию Алисы. Под ним находятся 277316 предыдущих блоков (в том числе блок #0), связанных друг с другом в цепь (blockchain) вплоть до нулевого, известного как блок *генезиса*. Со временем, по мере того как "высота" в блоках увеличивается, то же самое происходит и с вычислительной сложностью каждого блока и с цепочкой в целом. Блоки, добытые после того, который содержит транзакцию Алисы, становятся дополнительной гарантией по мере того, как растет количество вычислений и длина цепочки. По соглашению,

любой блок с более чем шестью подтверждениями считается безотзывным, так как для его отмены потребуется пересчитать шесть блоков, а это огромное количество вычислений. Мы рассмотрим процесс майнинга и вопрос доверительности более подробно в [\[ch8\]](#).

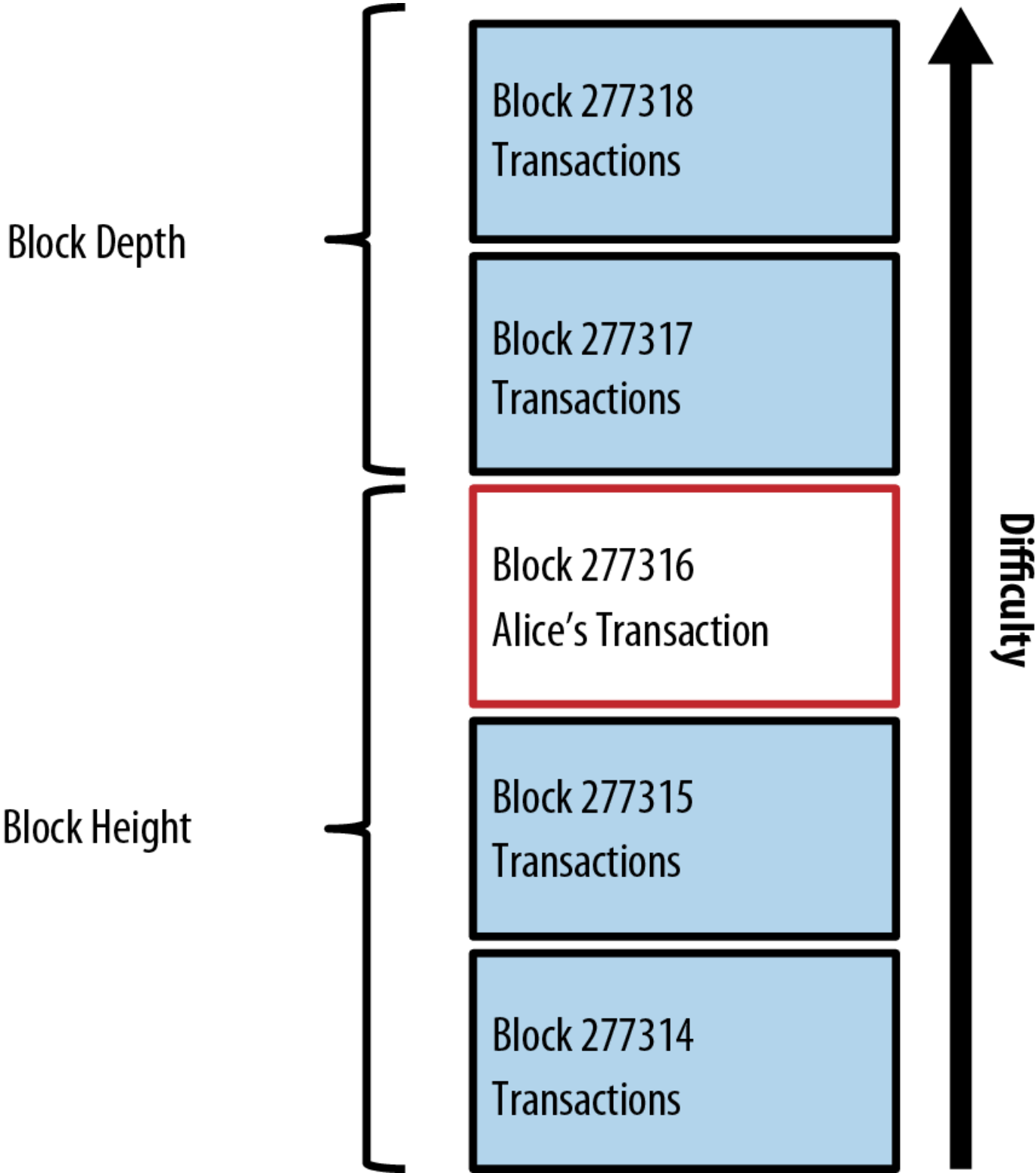


Figure 9. Танзакция Алисы, включенная в блок #277316

Растрачивание транзакции

Теперь, когда транзакция Алисы была встроена в блокчейн в качестве записи в блоке, она становится частью распределенной бухгалтерской книги Биткоин и видной для всех приложений Биткойн. Каждый отдельный Биткоин-клиент может самостоятельно проверить транзакцию на правильность и возможность потратить средства. Клиенты с полным индексом способны отследить источник средств с момента, когда они были созданы в блоке, последовательно транзакция к транзакции и так, пока они не достигнут адреса Боба. Легкие клиенты могут делать то, что называется упрощенной проверкой перевода (см. [\[spv_nodes\]](#)), подтверждая, что транзакция находится в блокчейне и после нее найдено несколько блоков.

Теперь Боб может потратить выход этой и других транзакций, создав свои собственные транзакции, которые бы ссылались на эти выходы в качестве входов. Например, Боб может заплатить подрядчику или поставщику путем передачи стоимости оплаты за чашку кофе Алисы этим новым владельцам. Скорее всего, программное обеспечение Биткоин Боба будет агрегировать множество мелких платежей в большой платеж, возможно, концентрируя доход целого дня в единую транзакцию. Это сведет различные платежи в один адрес, используемый в качестве "проверочного" счета магазина. Диаграмма агрегирующей транзакции изображена в [Транзакция слияния средств](#).

Так как Боб тратит средства, полученные от Алисы и других клиентов, он удлиняет цепочку транзакций, которые также добавляются в глобальный гроссбух и становятся видны всем. Давайте предположим, что Боб платит своему веб-дизайнеру Гопешу из Бангалоре за новый дизайн веб-сайта. Теперь цепочка транзакций будет выглядеть как здесь [Транзакция Алисы](#). [как часть цепи транзакций от Джо к Гопешу](#).

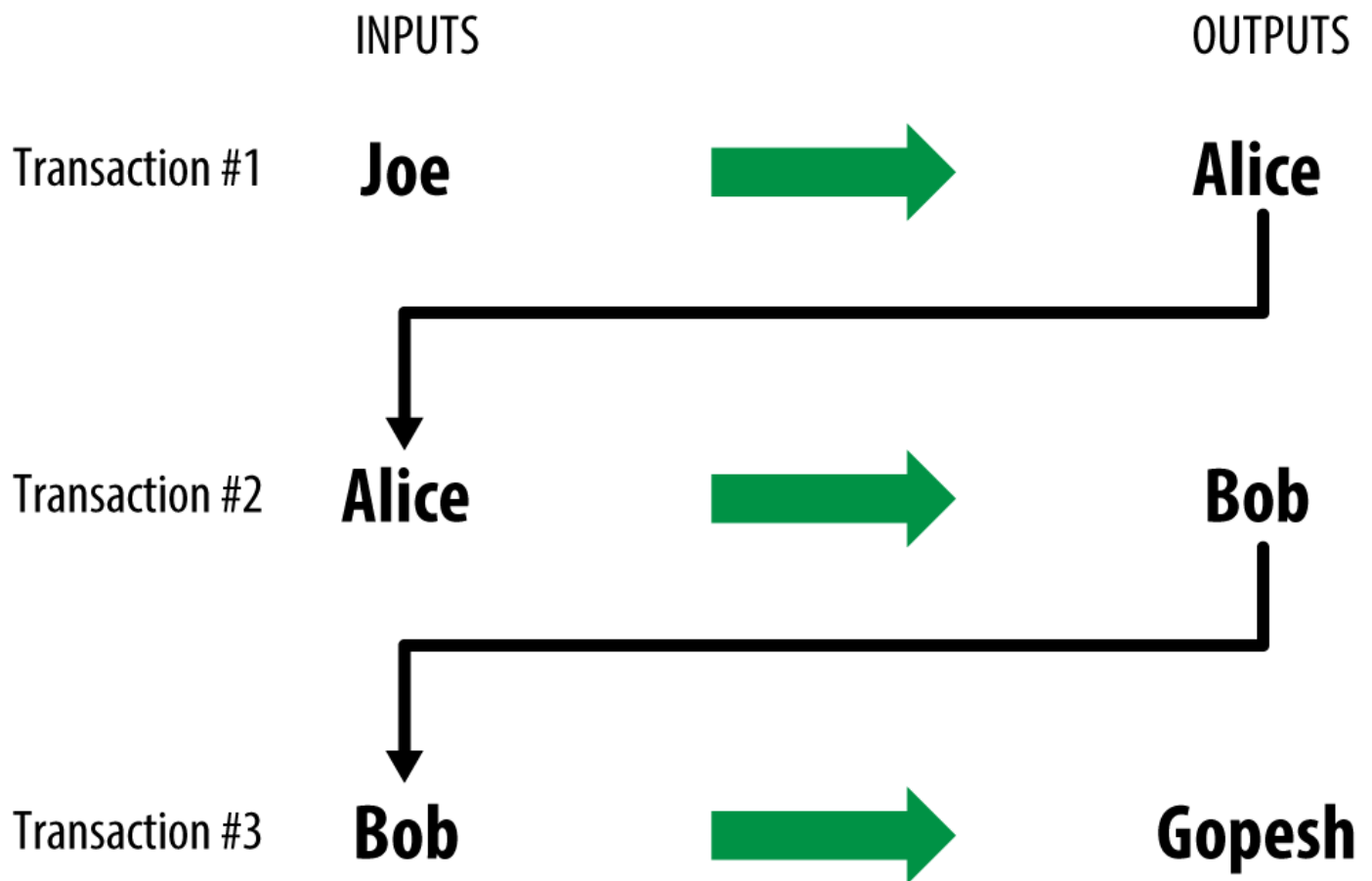


Figure 10. Транзакция Алисы, как часть цепи транзакций от Джо к Гопешу

Биткоин-клиент

Bitcoin Core: эталонная реализация

Вы можете скачать базовый клиент *Bitcoin Core*, также известный как "клиент Сатоши," с bitcoin.org. Базовый клиент реализует все аспекты системы Биткоин, в том числе кошелек, движок проверки транзакций с полной копией глобальной бухгалтерской книги blockchain, и полный сетевой узел пиринговой сети Биткоин.

Чтобы скачать базовый клиент, на странице [Bitcoin's Choose Your Wallet page](https://bitcoin.org/en/choose-your-wallet), выберите Bitcoin Core. Вы скачаете соответствующую вашей операционной системе программу установки. Для Windows, это либо архив ZIP, либо исполняемый файл.exe. Для Mac OS это образ диска .dmg. Версии для Linux включают в себя пакет PPA для Ubuntu или архив tar.gz. Странице [bitcoin.org](https://bitcoin.org/en/choose-your-wallet) со списком рекомендуемых Биткоин-клиентов показана на [Выбор Биткоин-клиента на bitcoin.org](#).

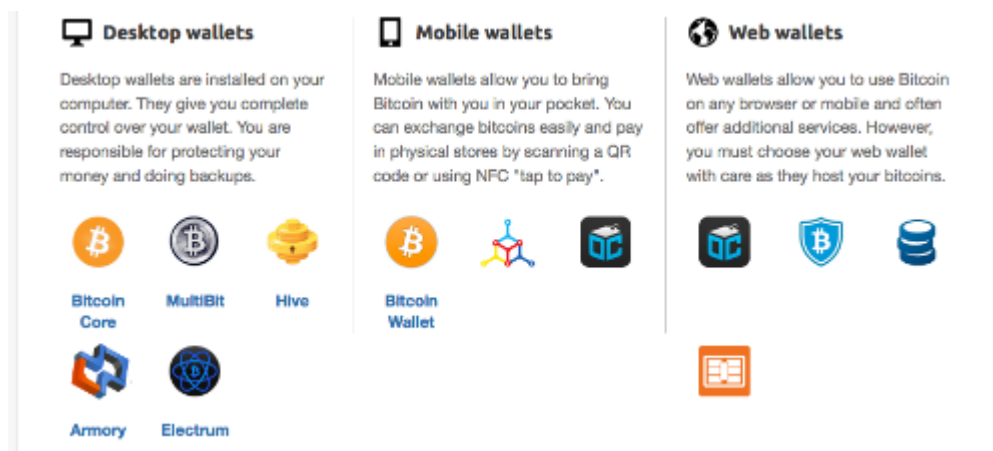


Figure 1. Выбор Биткоин-клиента на bitcoin.org

Первый запуск Bitcoin Core

Если вы скачаете установочный пакет, такой как .exe, .dmg, или PPA, вы можете установить его так же, как любое приложение вашей операционной системы. Для Windows, запустите .exe и следуйте пошаговым инструкциям. Для Mac OS, запустите .dmg и перетащите значок Bitcoin-Qt в папку *Applications*. Для Ubuntu, дважды щелкните PPA в Файловом Проводнике, и для установки откроется менеджер пакетов. После того, как вы завершили установку, в списке приложений у вас появится новое приложение под названием Bitcoin-Qt. Дважды щелкните значок, чтобы запустить Биткоин-клиент.

При первом запуске Bitcoin Core начнется загрузка blockchain, процесс, который может занять несколько дней (см [Экран Bitcoin Core во время инициализации блокчейна](#)). Оставьте его запущенным в фоновом режиме, пока не появится "Синхронизировано" рядом с балансом.

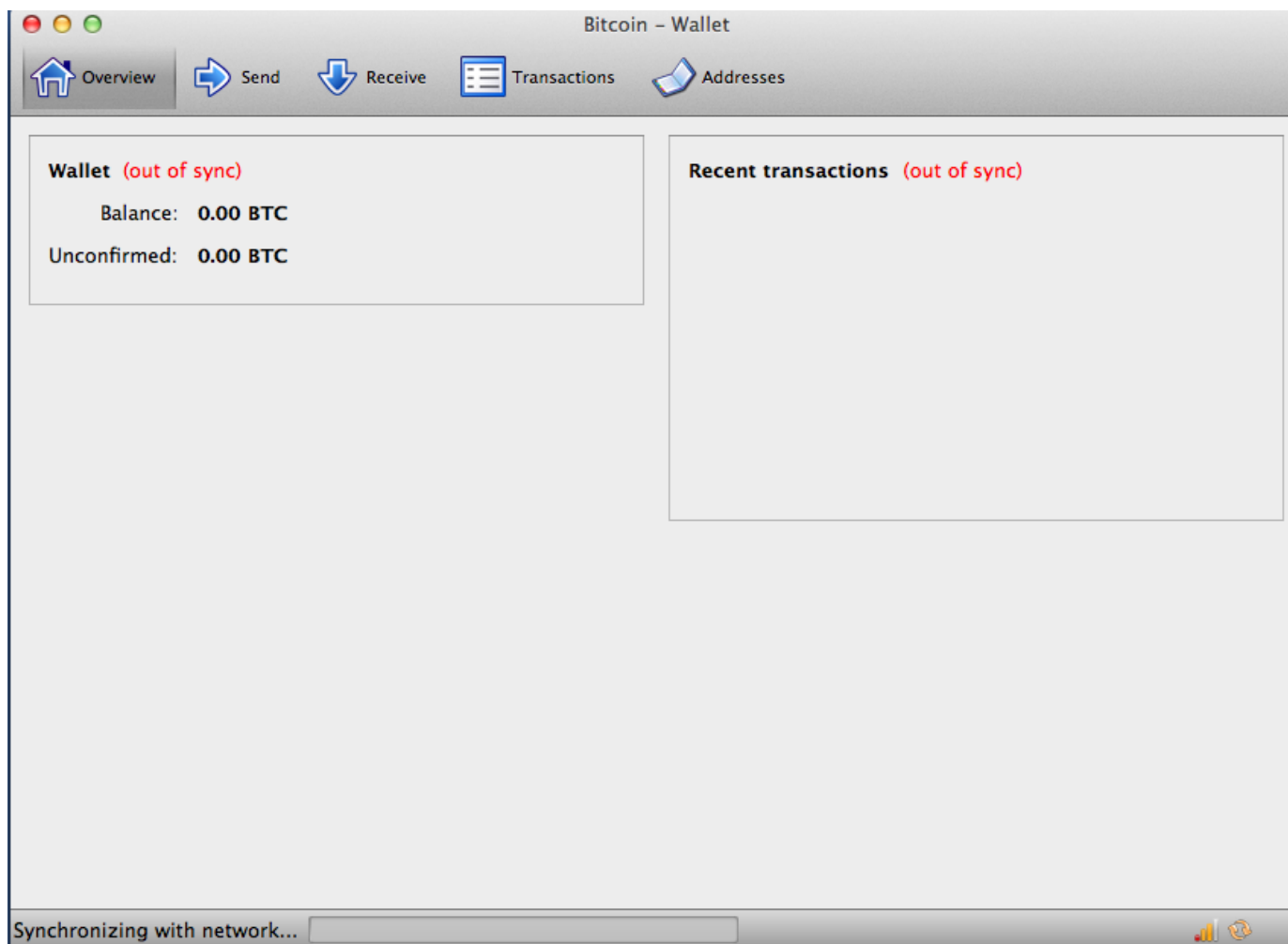


Figure 2. Экран Bitcoin Core во время инициализации блокчейна

ТИП

Bitcoin Core хранит полную копию книги транзакций (blockchain), с каждой транзакцией, которая когда-либо произошла в сети Биткоин с момента ее создания в 2009 году. Набор данных размером в несколько гигабайт (приблизительно 16 ГБ в конце 2013 года) загружается в течение нескольких дней. Клиент не сможет обрабатывать транзакции или обновить баланс счета пока blockchain полностью не загрузится. В течение этого времени, клиент будет отображать "нет синхронизации" рядом с балансами и "Синхронизаю" в нижней части окна. Убедитесь, что у вас есть достаточно дискового пространства, пропускной способности и времени, чтобы завершить начальную синхронизацию.

Сборка Bitcoin Core из исходных кодов

Для разработчиков, есть также возможность скачать полный исходный код в ZIP архиве или путем клонирования репозитория из GitHub. На странице [GitHub bitcoin page](#) выберите "Скачать ZIP" на боковой панели. В качестве альтернативы, используйте команду Git для получения локальной копии исходного кода в вашей системе. В следующем примере, мы клонируем исходный код из Unix-подобной командной строки в Linux или Mac OS:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 31864, done.
remote: Compressing objects: 100% (12007/12007), done.
remote: Total 31864 (delta 24480), reused 26530 (delta 19621)
Receiving objects: 100% (31864/31864), 18.47 MiB | 119 KiB/s, done.
Resolving deltas: 100% (24480/24480), done.
$
```

TIP

Эти инструкции и вывод команды могут отличаться от версии к версии. Следуйте документацию, которая поставляется с кодом, даже если она отличается от инструкций, которые вы видите здесь, и не удивляйтесь, если вывод на экране немного отличается от примеров здесь.

Когда операция клонирования git-репозитория завершена, вы получите полную локальную копию репозитория исходного кода в каталоге *bitcoin*. Перейдите в этот каталог, набрав `cd bitcoin` в командной строке:

```
$ cd bitcoin
```

По умолчанию локальная копия будет содержать самый свежий код, который может быть нестабильным или уровня бета-версии. Перед компиляцией кода, выберите конкретную версию, заглянув в *метки*. Это позволит синхронизировать локальную копию с конкретной временной меткой, определенной ключевым словом. Метки используются разработчиками, чтобы пометить конкретные релизы по номеру версии. Сначала, для просмотра доступных меток, мы используем команду `git tag`:

```
$ git tag
v0.1.5
v0.1.6test1
v0.2.0
v0.2.10
v0.2.11
v0.2.12

[... много других тегов ...]

v0.8.4rc2
v0.8.5
v0.8.6
v0.8.6rc1
v0.9.0rc1
```

Список тегов показывает все выпущенные версии Bitcoin. По соглашению, *релиз-кандидаты*, предназначены для тестирования, имеют суффикс "rc". Стабильные релизы не имеют никакого суффикса. Из предыдущего списка, выберите самую последнюю версию, которая на момент написания книги была v0.9.0rc1. Для синхронизации локального кода с этой версией, используйте команду `git checkout`:

```
$ git checkout v0.9.0rc1
Note: checking out 'v0.9.0rc1'.

HEAD is now at 15ec451... Merge pull request #3605
$
```

Исходный код включает в себя документацию. Основная документация может быть найдена в *README.md* в каталоге `bitcoin`. Наберите `more README.md` для просмотра. Используйте клавишу пробел для перемещения на следующую страницу. В этой главе, мы соберем Биткоин-клиент для командной строки, также известный как `bitcoind` под Linux. Просмотрите инструкции по компиляции `bitcoind` для вашей платформы, набрав `more doc/build-unix.md`. Альтернативные инструкции для Mac OS X и Windows, можно найти в каталоге *doc*, под именами *build-osx.md* или *build-msw.md* соответственно.

Внимательно изучите предварительные требования по сборке из исходных текстов, находящиеся в первой части документации. Там перечислены библиотеки, которые должны присутствовать в вашей системе, прежде чем вы сможете приступить к сборке ПО Биткоин. Если эти предварительные условия отсутствуют, процесс сборки завершится с ошибкой. Если ошибка сборки возникнет по причине отсутствия какой-либо требуемой библиотеки, то ее можно будет установить, а затем продолжить процесс сборки с места, где вы остановились. Предполагая, что предварительные требования удовлетворены, вы можете начать процесс сборки с генерации набора скриптов сборки с помощью скрипта *autogen.sh*.

TIP

Процедура сборки Bitcoin Core стала использовать `autogen/configure/make` начиная с версии 0.9. Старые версии использовали простой `Makefile`, поэтому следуйте инструкциям для версии, которую вы хотите скомпилировать. Система `autogen/configure/make`, введенная в 0,9, вероятно, останется системой сборки для всех последующих версий.

```
$ ./autogen.sh
configure.ac:12: installing `src/build-aux/config.guess'
configure.ac:12: installing `src/build-aux/config.sub'
configure.ac:37: installing `src/build-aux/install-sh'
configure.ac:37: installing `src/build-aux/missing'
src/Makefile.am: installing `src/build-aux/depcomp'
$
```

Скрипт *autogen.sh* создает набор автоматических скриптов конфигурации, которые определяют правильные настройки и определяют, что вы имеете все необходимые библиотеки для компиляции кода. Наиболее важным из них является скрипт *configure*, который предлагает целый ряд различных опций для настройки процесса сборки. Наберите *./configure --help* для просмотра опций:

```
$ ./configure --help

'configure' configures Bitcoin Core 0.9.0 to adapt to many kinds of systems.

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

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE. See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help                display this help and exit
    --help=short            display options specific to this package
    --help=recursive        display the short help of all the included packages
  -V, --version             display version information and exit

[... множество других опций и переменных ...]

Optional Features:
  --disable-option-checking ignore unrecognized --enable/--with options
  --disable-FEATURE        do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]   include FEATURE [ARG=yes]

[... еще опции ...]

Используйте эти переменные, чтобы переопределить выбор, сделанный 'configure'
или для того, чтобы
помочь ей найти библиотеки и программы с нестандартными названиями
/расположением.

Отчеты об ошибках можно посылать на адрес <info@bitcoin.org>.

$
```

Скрипт *configure* позволяет включить или выключить определенные функции *bitcoind* через использование флагов *--enable-FEATURE* и *--disable-FEATURE* флаги, где *FEATURE* заменяется названием функции, как указано в выводе справки. В этой главе, мы соберем *bitcoind* со всеми опциями по умолчанию. Мы не будем использовать флаги конфигурации, но вам следует просмотреть их, чтобы понять какие функции являются частью клиента. Далее, запустите

скрипт `configure` для автоматического обнаружения всех необходимых библиотек и создания сценария сборки для вашей 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... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes

[... проверка различных других функций системы ...]

configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating src/test/Makefile
config.status: creating src/qt/Makefile
config.status: creating src/qt/test/Makefile
config.status: creating share/setup.nsi
config.status: creating share/qt/Info.plist
config.status: creating qa/pull-tester/run-bitcoind-for-test.sh
config.status: creating qa/pull-tester/build-tests.sh
config.status: creating src/bitcoin-config.h
config.status: executing depfiles commands
$
```

Если все пойдет хорошо, команда `configure` завершится созданием скриптов для сборки `bitcoind`. Если какие-либо библиотеки не будут найдены или обнаружатся ошибки, команда `configure` завершится с ошибкой, вместо создания сценариев сборки. Если происходит ошибка, то, это скорее всего, из-за отсутствия или несовместимости определенной библиотеки. Просмотрите документацию по сборке еще раз и убедитесь, что вы установили все необходимые зависимости. После этого запустите `configure` еще раз и посмотрите, не пропала ли ошибка. Далее, произведите компиляцию из исходного кода. Этот процесс может занять до часа. В процессе компиляции вы увидите диагностические сообщения каждые несколько секунд или минут либо ошибку, в случае, если что-то пойдет не так. Если процесс компиляции прервется, то возобновить его можно в любое время. Введите `make` для того, чтобы начать компиляцию:

```
$ make
Making all in src
make[1]: Entering directory `/home/ubuntu/bitcoin/src'
make all-recursive
make[2]: Entering directory `/home/ubuntu/bitcoin/src'
Making all in .
make[3]: Entering directory `/home/ubuntu/bitcoin/src'
CXX      addrman.o
CXX      alert.o
CXX      rpcserver.o
CXX      bloom.o
CXX      chainparams.o

[... различные другие сообщения компилятора ...]

CXX      test_bitcoin-wallet_tests.o
CXX      test_bitcoin-rpc_wallet_tests.o
CXXLD    test_bitcoin
make[4]: Leaving directory `/home/ubuntu/bitcoin/src/test'
make[3]: Leaving directory `/home/ubuntu/bitcoin/src/test'
make[2]: Leaving directory `/home/ubuntu/bitcoin/src'
make[1]: Leaving directory `/home/ubuntu/bitcoin/src'
make[1]: Entering directory `/home/ubuntu/bitcoin'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/home/ubuntu/bitcoin'
$
```

Если все прошло удачно, bitcoind скомпилировался. Последним шагом является установка bitcoind в системный каталог при помощи команды make:

```
$ sudo make install
Making install in src
Making install in .
/bin/mkdir -p '/usr/local/bin'
/usr/bin/install -c bitcoind bitcoin-cli '/usr/local/bin'
Making install in test
make install-am
/bin/mkdir -p '/usr/local/bin'
/usr/bin/install -c test_bitcoin '/usr/local/bin'
$
```

Вы можете убедиться, что Bitcoin установлен корректно, запросив у ОС пути до двух исполняемых файлов:

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

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

По умолчанию установщик копирует их в `/usr/local/bin`. При первом запуске `bitcoind`, он будет напоминать вам создать конфигурационный файл с надежным паролем для интерфейса JSON-RPC. Запустите `bitcoind` набрав `bitcoind` в терминале:

```
$ bitcoind
Error: To use the "-server" option, you must set a rpcpassword in the configuration file:
/home/ubuntu/.bitcoin/bitcoin.conf
It is recommended you use the following random password:
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(вам не нужно запоминать этот пароль)
Имя пользователя и пароль не должны быть одинаковыми.
Если файл не существует, создайте его с правами только для чтения владельцем.
Рекомендуется также установить alertnotify, чтобы получать уведомления о
проблемах;
например: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

Отредактируйте файл конфигурации в своем любимом редакторе и установите параметры, сразу поменяв пароль в соответствии с рекомендациями `bitcoind`. *Не* используйте пароль, показанный здесь. Создайте файл внутри каталога `.bitcoin` так, чтобы он назывался `.bitcoin/bitcoin.conf` и введите имя пользователя и пароль:

```
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
```

Редактируя файл конфигурации, вы, возможно, захотите установить несколько других опций, например `txindex` (см. [Индекс БД Транзакций и опция txindex](#)). Для получения полного списка доступных опций, наберите `bitcoind --help`.

Теперь запустите клиент Bitcoin Core. После первого запуска он начнет скачивать `blockchain` из сети. Это файл размером в несколько гигабайт, и его скачивание займет в среднем два дня. Вы можете сократить время инициализации `blockchain`, загрузив частичную копию `blockchain`, используя клиент BitTorrent с [SourceForge](#).

Запустите `bitcoind` в фоновом режиме при помощи опции `-daemon`:

```
$ bitcoind -daemon

Bitcoin version v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
Using OpenSSL version OpenSSL 1.0.1c 10 May 2012
Default data directory /home/bitcoin/.bitcoin
Using data directory /bitcoin/
Using at most 4 connections (1024 file descriptors available)
init message: Verifying wallet...
dbenv.open LogDir=/bitcoin/database ErrorFile=/bitcoin/db.log
Bound to [::]:8333
Bound to 0.0.0.0:8333
init message: Loading block index...
Opening LevelDB in /bitcoin/blocks/index
Opened LevelDB successfully
Opening LevelDB in /bitcoin/chainstate
Opened LevelDB successfully

[... различные другие сообщения ...]
```

Использование JSON-RPC интерфейса Bitcoin Core из командной строки

Клиент Bitcoin Core реализует интерфейс JSON-RPC, к которому можно получить доступ с помощью утилиты командной строки `bitcoin-cli`. Из командной строки можно экспериментировать в интерактивном режиме с возможностями, которые также доступны программно через API. Для того, чтобы увидеть список доступных команд RPC, вызовите команду `help`:

```
$ 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"
decodescript "hex"
dumpprivkey "bitcoinaddress"
dumpwallet "filename"
getaccount "bitcoinaddress"
getaccountaddress "account"
getaddednodeinfo dns ( "node" )
getaddressesbyaccount "account"
getbalance ( "account" minconf )
getbestblockhash
```

```

getblock "hash" ( verbose )
getblockchaininfo
getblockcount
getblockhash index
getblocktemplate ( "jsonrequestobject" )
getconnectioncount
getdifficulty
getgenerate
gethashespersec
getinfo
getmininginfo
getnettotals
getnetworkhashps ( blocks height )
getnetworkinfo
getnewaddress ( "account" )
getpeerinfo
getrawchangeaddress
getrawmempool ( verbose )
getrawtransaction "txid" ( verbose )
getreceivedbyaccount "account" ( minconf )
getreceivedbyaddress "bitcoinaddress" ( minconf )
gettransaction "txid"
gettxout "txid" n ( includemempool )
gettxoutsetinfo
getunconfirmedbalance
getwalletinfo
getwork ( "data" )
help ( "command" )
importprivkey "bitcoinprivkey" ( "label" rescan )
importwallet "filename"
keypoolrefill ( newsize )
listaccounts ( minconf )
listaddressgroupings
listlockunspent
listreceivedbyaccount ( minconf includeempty )
listreceivedbyaddress ( minconf includeempty )
listsinceblock ( "blockhash" target-confirmations )
listtransactions ( "account" count from )
listunspent ( minconf maxconf ["address",...] )
lockunspent unlock [{"txid":"txid","vout":n},...]
move "fromaccount" "toaccount" amount ( minconf "comment" )
ping
sendfrom "fromaccount" "tobitcoinaddress" amount ( minconf "comment" "comment-to" )
sendmany "fromaccount" {"address":amount,...} ( minconf "comment" )
sendrawtransaction "hexstring" ( allowhighfees )
sendtoaddress "bitcoinaddress" amount ( "comment" "comment-to" )
setaccount "bitcoinaddress" "account"
setgenerate generate ( genproclimit )

```

```
settxfee amount
signmessage "bitcoinaddress" "message"
signrawtransaction "hexstring" (
[{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...]
["privatekey1",...] sighashtype )
stop
submitblock "hexdata" ( "jsonparametersobject" )
validateaddress "bitcoinaddress"
verifychain ( checklevel numblocks )
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

Получение информации о статусе клиента Bitcoin Core

Команды: getinfo

RPC-команда getinfo показывает основную информацию о состоянии данного узла, кошелька, и базы данных блокчейн. Ее можно вызвать с помощью bitcoin-cli:

```
$ bitcoin-cli getinfo
```

```
{
  "version" : 90000,
  "protocolversion" : 70002,
  "walletversion" : 60000,
  "balance" : 0.00000000,
  "blocks" : 286216,
  "timeoffset" : -72,
  "connections" : 4,
  "proxy" : "",
  "difficulty" : 2621404453.06461525,
  "testnet" : false,
  "keypoololdest" : 1374553827,
  "keypoolsize" : 101,
  "paytxfee" : 0.00000000,
  "errors" : ""
}
```

Данные возвращаются в формате JavaScript Object Notation (JSON), который может быть легко считан при помощи программы на любом языке программирования, а также весьма удобен для восприятия человеком. Среди этих данных мы видим номера версий ПО клиента (90000), протокола (70002), и кошелька (60000). Мы также видим, что текущий баланс, содержащийся в

кошельке равен нулю. Мы видим текущую высоту блока известную этому клиенту (286216). Мы также видим, различные статистические данные о сети Биткоин и параметры, связанные с этим клиентом. Мы изучим эти параметры более подробно в оставшейся части этой главы.

ТИП

Для того, чтобы bitcoind смог "догнать" блокчейн до его текущей высоты понадобится некоторое время, возможно больше, чем день, так как придется получить блоки от других клиентов сети. Вы можете проверить свой прогресс с помощью getinfo и увидеть количество известных блоков.

Настройка кошелька и шифрования

Команды: encryptwallet, walletpassphrase

Прежде чем приступить к созданию ключей и запуску других команд, вы должны сначала зашифровать кошелек паролем. В нашем примере мы используем команду encryptwallet с паролем "foo". Очевидно вы захотите заменить "foo" более стойким паролем!

```
$ bitcoin-cli encryptwallet foo
wallet encrypted; Bitcoin server stopping, restart to run with encrypted wallet. The
keypool has been flushed, you need to make a new backup.
$
```

Вы можете убедиться, что кошелек теперь зашифрован, снова запустив команду getinfo. На этот раз вы увидите новую запись под названием unlocked_until. Это счетчик, показывающий, как долго пароль расшифровки кошелька будет храниться в памяти, сохраняя кошелек в разблокированном состоянии. Сначала это значение будет равно нулю, что будет означать, что кошелек заблокирован:

```
$ bitcoin-cli getinfo
```

```
{
  "version" : 90000,
  #[...] другая информация...

  "unlocked_until" : 0,
  "errors" : ""
}
$
```

Чтобы разблокировать кошелек, дайте команду walletpassphrase, которая принимает два параметра — пароль и количество секунд до тех пор, пока бумажник снова автоматически не заблокируется (таймер):

```
$ bitcoin-cli walletpassphrase foo 360
$
```

Вы можете подтвердить разблокировку кошелька и проверить таймаут, запустив `getinfo` снова:

```
$ bitcoin-cli getinfo
```

```
{
  "version" : 90000,
  #[...] другая информация ...]
  "unlocked_until" : 1392580909,
  "errors" : ""
}
```

Резервное копирование кошелька, простой текстовый дамп, и восстановление

Команды: `backupwallet`, `importwallet`, `dumpwallet`

Далее, мы попрактикуемся в создании резервной копии кошелька, а затем восстановим кошелек из резервной копии. Используйте команду `backupwallet` для резервного копирования, передав имя файла в качестве параметра. Здесь мы делаем резервное копирование бумажника в файл *wallet.backup*:

```
$ bitcoin-cli backupwallet wallet.backup
$
```

Теперь, для восстановления резервной копии, используйте команду `importwallet`. Если ваш кошелек заблокирован, вам понадобится его сначала разблокировать (см `walletpassphrase` в предыдущем разделе):

```
$ bitcoin-cli importwallet wallet.backup
$
```

Команда `dumpwallet` может быть использована для того, чтобы сдампить кошелек в человеко-читаемый текстовый файл:

```
$ bitcoin-cli dumpwallet wallet.txt
$ more wallet.txt
# Wallet dump created by Bitcoin v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
# * Created on 2014-02- 8dT20:34:55Z
# * Best block at time of backup was 286234
(0000000000000000f74f0bc9d3c186267bc45c7b91c49a0386538ac24c0d3a44),
#   mined on 2014-02- 8dT20:24:01Z

KzTg2wn6Z8s7ai5NA9MVX4vstHRsqP26QKJCzLg4JvFrp6mMaGB9 2013-07- 4dT04:30:27Z change=1 #
addr=16pJ6XkwSQv5ma5FSXMRPaXEYrENCEg47F
Kz3dVz7R6mUpXzdZy4gJEVZxXJwA15f198eVui4CUivXotzLBDKY 2013-07- 4dT04:30:27Z change=1 #
addr=17oJds8kaN8LP8kuAkWTco6ZM7BGXFC3gk
[... много других ключей ...]

$
```

Адреса кошелька и получение транзакций

Команды: `getnewaddress`, `getreceivedbyaddress`, `listtransactions`, `getaddressesbyaccount`, `getbalance`

Базовый Биткоин-клиент создает пул адресов, размер которого показан в поле `keypoolsize` в выводе команды `getinfo`. Эти адреса генерируются автоматически и могут быть использованы в качестве публичных адресов для приема платежей или адресов для сдачи. Чтобы получить один из этих адресов, используйте команду `getnewaddress`:

```
$ bitcoin-cli getnewaddress
1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
```

Теперь мы можем использовать этот адрес для отправки небольшого количества биткоинов на наш `bitcoind`-кошелек со внешнего кошелька (подразумевая, что у вас есть немного биткоинов на бирже, в веб-кошельке, или в другом `bitcoind`-кошельке). Для этого примера, мы пошлем 50 millibits (0.050 Bitcoin) на адрес, полученный выше.

Теперь мы можем запросить у `bitcoind` клиента сумму, полученную по этому адресу, и указать необходимое количество подтверждений для того, чтобы сумма считалась на балансе. Для этого примера, мы укажем нуль подтверждений. Через несколько секунд после отправки биткоинов из другого кошелька, мы увидим, что это количество теперь отражено в нашем бумажнике. Это делается с помощью команды `getreceivedbyaddress` с адресом и количеством подтверждений установленных в нуль (0) в качестве аргументов:

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL 0
0.05000000
```

Если мы опустим нуль в конце этой команды, мы увидим только суммы, которые имеют, по крайней мере `minconf` подтверждений, где `minconf`—это настройка, обозначающая минимальное число подтверждений, прежде чем сделка будет зачислена в баланс. Настройка `minconf` прописана в файле конфигурации `bitcoind`. Поскольку транзакция отправки была создана лишь в несколько секунд назад, она до сих пор не подтверждена, и поэтому мы увидим нулевой баланс:

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
0.00000000
```

Транзакции, полученные всем кошельком также могут быть отображены с помощью команды `listtransactions`:

```
$ bitcoin-cli listtransactions
```

```
[
  {
    "account" : "",
    "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
    "category" : "receive",
    "amount" : 0.05000000,
    "confirmations" : 0,
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908
  }
]
```

Получить список всех адресов кошелька можно, используя команду `getaddressesbyaccount`:

```
$ bitcoin-cli getaddressesbyaccount ""
```

```
[
  "1LQoTPYy1TyERbNV4zZbhEmgyfAipC6eqL",
  "17vrg8uwMQUibkvS2ECRX4zpcVJ78iFaZS",
  "1FvRHWWhHBBZA8cGRRsGiAeqEzUmjJkJQWR",
  "1NVJK3JJsL41BF1KyxruYJW5XHjunjfp2jz",
  "14MZqqzCxjc99M5ipsQSRfieT7qPZcM7Df",
  "1BhrGvtKFjTAhGdPGbrEwP3xvFjkJBuFCa",
  "15nem8CX91XtQE8B1Hdv97jE8X44H3DQMT",
  "1Q3q6taTsUiv3mMemEuQQJ9sGLEGaSjo81",
  "1HoSiTg8sb16oE6SrmazQEwcGEv8obv9ns",
  "13fE8BGhBvnoy68yZKuWJ2hheYKovSDjqM",
  "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
  "1KHUmVfCJteJ21LmRXHSpPoe23rXKifAb2",
  "1LqJZz1D9yHxG4cLkdujngG5jNNGmPeAMD"
]
```

Наконец, команда `getbalance` показывает общий баланс кошелька, сложив все транзакции, имеющие не менее `minconf` подтверждений:

```
$ bitcoin-cli getbalance
0.05000000
```

ТИП

Если транзакция еще не подтверждена, баланс возвращенный командой `getbalance` будет равен нулю. Опция конфигурации `"minconf"` определяет минимальное количество подтверждений, необходимых, прежде чем сделка покажется в балансе.

Исследование и декодирование транзакций

Команды: `gettransaction`, `getrawtransaction`, `decoderawtransaction`

Теперь изучим подробнее поступившую транзакцию, которую мы увидели в выводе команды `gettransaction` ранее. Мы можем получить транзакцию по ее хэшу, показанному в поле `txid` вывода команды `gettransaction`:

```
{
  "amount" : 0.05000000,
  "confirmations" : 0,
  "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
  "time" : 1392660908,
  "timereceived" : 1392660908,
  "details" : [
    {
      "account" : "",
      "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
      "category" : "receive",
      "amount" : 0.05000000
    }
  ]
}
```

TIP

Идентификаторы транзакций не заслуживают доверия, пока транзакция не будет подтверждена. Отсутствие хэша транзакции в блокчейне не означает, что транзакция не была обработана. Это явление известно как "пластичность транзакции", так как хэши транзакций могут быть изменены до подтверждения в блоке. После подтверждения txid неизменен и заслуживает доверия.

Вид транзакции в выводе команды `gettransaction` является упрощенной формой. Для того, чтобы получить полный код транзакции и декодировать его, мы используем две команды: `getrawtransaction` и `decoderawtransaction`. Во-первых, `getrawtransaction` принимает *хэш транзакции (txid)* в качестве параметра и возвращает полную транзакцию в виде "сырой" шестнадцатеричной строки, в точности, как она существует в сети Биткоин:

Для декодирования этой шестнадцатеричной строки, используется команда `decoderawtransaction`. Скопируйте и вставьте эту строку в качестве первого параметра `decoderawtransaction` для того, чтобы получить полное содержимое в виде структуры данных JSON (по причинам форматирования шестнадцатеричная строка в следующем примере укорочена):

После декодирования становятся видны все компоненты этой транзакции, в том числе входы и выходы. В этом случае мы видим, что в транзакции, зачислившей 50 millibit на наш новый адрес, используется один вход и два выхода. Входом этой транзакции становится выход из ранее подтвержденной транзакции (показан как `vin txid`, начинающийся с `d3c7`). Два выхода соответствуют зачислению 50 millibit и выходу со сдачи назад к отправителю.

Мы можем далее исследовать блокчейн путем изучения предыдущей транзакции, на которую ссылается ее `txid` в этой транзакции, используя те же команды (т.е. `gettransaction`). Перемещаясь от транзакции к транзакции, мы можем проследить цепочку сделок назад во времени и пронаблюдать как средства передавались от владельца владельцу.

Как только транзакция, которую мы получили, была подтверждена путем включения в блок, команда `gettransaction` вернет дополнительную информацию, показывающую *хэш (идентификатор) блока*, в который сделка была включена:

Здесь мы видим новую информацию в записях `blockhash` (хэш блока, в который была включена транзакция) и `blockindex` со значением 18 (указывает, что наша транзакция была 18-ой сделкой в этом блоке).

Индекс БД Транзакций и опция `txindex`

По умолчанию, Bitcoin Core строит базу данных, содержащую *только* транзакции, относящиеся к кошельку пользователя. Если вы хотите иметь возможность доступа к *любой* транзакции при помощи команд наподобие `gettransaction`, вам придется сконфигурировать Bitcoin Core построить полный индекс транзакций. Этого можно достичь при помощи опции `txindex`. Установите `txindex=1` в файле конфигурации Bitcoin Core (обычно находящийся в вашем домашнем каталоге в `.bitcoin/bitcoin.conf`). Как только вы поменяете этот параметр, вам понадобится перезапустить `bitcoind` и дождаться, пока тот перестроит индекс.

Исследование блоков

Команды: `getblock`, `getblockhash`

Теперь, когда мы знаем, в какой блок попала наша транзакция, мы можем запросить этот блок. Используем команду `getblock` с хешем блока в качестве параметра:

Блок содержит 367 транзакций и, как видно, 18-ая сделка в списке с `txid 9ca8f9...` как раз та, в которой мы послали 50 millibits на наш адрес. Поле высота (`height`) говорит нам, что это 286384-ый блок в `blockchain`.

Мы можем также получить блок по его высоте, используя команду `getblockhash`, которая принимает высоту блока в качестве параметра и возвращает хэш блока:

Ниже мы получаем хэш блока генезиса, т.е. самого первого блока, найденного Сатоши Накамото, и имеющего высоту ноль:

Команды `getblock`, `getblockhash`, и `gettransaction` могут быть использованы для исследования блокчейна программно.

Создание, подпись и отсылка транзакций на основе проход: `<phrase role="keep-together">Неизрасходованных выходов</phrase>`

Команды: `listunspent`, `gettxout`, `createrawtransaction`, `decoderawtransaction`, `signrawtransaction`, `sendrawtransaction`

Транзакции в Биткоин основаны на концепте траты "выходов", которые получаются в результате предыдущих транзакций, создавая, таким образом, цепочку передачи прав от адреса к адресу. Наш кошелек получил транзакцию, которая присвоила один подобный выход нашему адресу. Как только эта операция будет подтверждена, мы сможем потратить выход.

Вначале мы используем команду `listunspent`, чтобы показать все неизрасходованные *подтвержденные* выходы в нашем кошельке:

```
$ bitcoin-cli listunspent
```

Мы видим, что сделка `9ca8f9...` создала выход (с порядковым номером `vout 0`), назначающей адресу `1hvfSo...` сумму в 50 millibits, которая к этому моменту получила семь подтверждений. В транзакциях в качестве входов используются ранее созданные выходы, на которые указывает ссылка предыдущих `txid` и порядковый номер `vout`. Далее мы создадим транзакцию, которая потратит 0-й `vout` из `txid 9ca8f9...` в качестве входа и назначит его на новый выход, который переведет значение на новый адрес.

Сначала давайте посмотрим на конкретный вывод более подробно. Используем `gettxout` для того, чтобы получить подробную информацию о данном неизрасходованном выходе. Выходы транзакции всегда могут быть найдены по `txid` and `vout` и эти параметры мы передаем `gettxout`:

Мы видим выход, который назначил 50 millibits нашему адресу `1hvf...`. Чтобы потратить этот вывод, мы создадим новую транзакцию. Во-первых, давайте создадим адрес, на который пошлем деньги:

```
$ bitcoin-cli getnewaddress  
1LnFTndy3qzXGN19Jwscj1T8LR3MVe3JDb
```

Мы пошлем 25 millibits по новому адресу `1LnFTn...`, который мы только что создали в нашем кошельке. В нашей новой транзакции, мы потратим 50 millibit выхода и отправим 25 millibits по этому новому адресу. Так как мы собираемся потратить выход предыдущей транзакции целиком, нам понадобится создать выход для сдачи. Мы создадим выход для сдачи обратно на адрес `1hvf...`, послав разницу обратно по адресу, с которого пришли основные средства. Наконец, нам также придется заплатить комиссионные сборы за эту транзакцию. Для того, чтобы оплатить комиссионные, мы будем уменьшим выход сдачи на 0,5 millibits, и вернем себе 24,5 millibits. Разница между суммой новых выходов ($25 \text{ mBTC} + 24,5 \text{ mBTC} = 49,5 \text{ mBTC}$) и входом (50 mBTC) будет собрана майнерами в качестве комиссионных за транзакцию.

Для создания этой транзакции используем `createrawtransaction`. В качестве параметров для `createrawtransaction` мы предоставляем вход транзакции (50 millibit с неизрасходованного выхода из нашей подтвержденной транзакции) и два выхода транзакции (деньги отправляем на новый адрес, а сдачу обратно на предыдущий адрес):

Команда `createrawtransaction` выдает сырую шестнадцатеричную строку, которая кодирует

транзакцию. Давайте убедимся, что все правильно, декодировав эту сырую строку обратно используя команду `decoderawtransaction`:

Это выглядит правильно! Наша новая транзакция "потребляет" неизрасходованный выход из нашей подтвержденной транзакции, а затем тратит его в двух выходах: один на 25 millibits на наш новый адрес и один для 24.5 millibits для сдачи на оригинальный адрес. Разница в 0.5 millibits представляет комиссионные и будет зачислена майнеру, который найдет блок и включит в него нашу транзакцию.

Как вы могли заметить, транзакция содержит пустой `scriptSig`, так как мы ее еще не подписали. Без подписи транзакция не имеет смысла; мы пока еще не доказали, что *владеем* адресом, из которого используется неизрасходованный выход. Подписывая, мы снимаем блокировку на выходе и доказываем права владения выходом, а значит можем потратить. Для того, чтобы подписать транзакцию мы используем команду `signrawtransaction`. Она принимает сырую шестнадцатеричную строку транзакции в качестве параметра:

ТИП

Зашифрованный кошелек должен быть разблокирован перед тем, как транзакция будет подписана, т.к. для подписывания требуется доступ к приватным ключам.

Команда `signrawtransaction` возвращает другую сырую транзакцию в формате шестнадцатеричной строки. Декодируем ее с помощью команды `decoderawtransaction` для того, чтобы посмотреть что изменилось:

Итак, входы, используемые в транзакции содержат `scriptSig`, который является цифровой подписью, подтверждающий право адреса `1hvz...` и разблокирующее выход так, что он может быть потрачен. Подпись делает эту сделку проверяемой любым узлом в сети Bitcoin.

Теперь пришло время отправить вновь созданную транзакцию в сеть. Мы можем это сделать при помощи команды `sendrawtransaction`, которая принимает сырую шестнадцатеричную строку, полученной при помощи `signrawtransaction`. Это та же самая строка, которую мы только что декодировали:

Команда `sendrawtransaction` возвращает *хэш транзакции* (`txid`) сразу после отсылки транзакции в сеть. Теперь мы можем запросить ID этой транзакции с помощью `gettransaction`:

```

{
  "amount" : 0.00000000,
  "fee" : -0.00050000,
  "confirmations" : 0,
  "txid" : "ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346",
  "time" : 1392666702,
  "timereceived" : 1392666702,
  "details" : [
    {
      "account" : "",
      "address" : "1LnFTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
      "category" : "send",
      "amount" : -0.02500000,
      "fee" : -0.00050000
    },
    {
      "account" : "",
      "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
      "category" : "send",
      "amount" : -0.02450000,
      "fee" : -0.00050000
    },
    {
      "account" : "",
      "address" : "1LnFTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
      "category" : "receive",
      "amount" : 0.02500000
    },
    {
      "account" : "",
      "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
      "category" : "receive",
      "amount" : 0.02450000
    }
  ]
}

```

Как и прежде, мы можем посмотреть подробности при помощи команд `getrawtransaction` и `decodetransaction`. Эти команды вернут ту же самую шестнадцатеричную строку, которую мы создали и декодировали ранее до того, как послать ее в сеть.

Альтернативные клиенты, библиотеки и инструменты

Кроме базового клиента (`bitcoind`), для взаимодействия с сетью и структурами данных Биткоин,

могут использоваться другие клиенты и библиотеки. Они реализованы на многих языках программирования.

Альтернативные имплементации включают в себя:

libbitcoin

Кросс-платформенная библиотека для C++

bitcoin explorer

Инструмент командной строки

bitcoin server

Полный узел и сервер

bitcoinj

Клиентская библиотека на Java

btcd

Полный узел на языке Go

Bits of Proof (BOP)

Реализация на Java уровня предприятия

picocoin

Легкая клиентская библиотека на C

pybitcointools

Библиотека для языка Python

pycoin

Еще одна библиотека для Python

Существует множество библиотек на различных других языках программирования и все время создаются новые.

Libbitcoin и Bitcoin Explorer

На основе кроссплатформенной библиотеки ("libraries, alternative", "libbitcoin library" libbitcoin написана реализация полного узла libbitcoin-server и утилита командной строки Bitcoin Explorer (bx).

Команды утилиты bx предлагают большинство возможностей, которые предоставляют команды bitcoind, проиллюстрированные в этой главе. Команды bx также предлагают некоторые инструменты управления и манипулирования ключами, которые не предлагаются bitcoind, в том числе детерминированные ключи type-2 и мнемоническое кодирование ключей, а также стелс-адреса, поддержку запросов.

Установка Bitcoin Explorer

Для того, чтобы начать использовать Bitcoin Explorer просто [download](#) скачайте [подписанный исполняемый для вашей операционной системы](#). Билды доступны для mainnet и testnet для Linux, OS X и Windows.

Введите `bx` без параметров, чтобы отобразить список всех доступных команд (см. [\[appdx_bx\]](#)).

Bitcoin Explorer также содержит инсталлятор для [building сборки из исходников под Linux и OS X](#), а также [проекта Visual Studio для Windows](#). С помощью Autotools утилиту можно собрать из исходников вручную. Библиотека `libbitcoin` будет установлена в качестве зависимости.

TIP	Bitcoin Explorer предлагает много полезных команд для кодирования и декодирования адресов, преобразования различных форматов и представлений, например Base16 (hex), Base58, Base58Check, Base64 и т.д.
------------	---

Установка Libbitcoin

Библиотека `libbitcoin` предоставляет инсталлятор для [building из исходников для Linux и OS X](#), а также [проект для Visual Studio](#). Исходные тексты также могут быть собраны вручную с помощью Autotools.

TIP	Установщик Bitcoin Explorer устанавливает как <code>bx</code> , так и библиотеку <code>libbitcoin</code> , так что если вы собрали <code>bx</code> из исходных текстов, вы можете пропустить этот шаг.
------------	--

python

Python-библиотека [python](#) написана и поддерживается Ричардом Киссом. Python поддерживает манипуляции с ключами Биткоин и транзакциями, и даже поддерживает язык сценариев на достаточном уровне, чтобы разбирать нестандартные сделки.

Библиотека `python` поддерживает Python 2 (2.7.x) и Python 3 (после 3.3), и поставляется с некоторыми полезными утилитами командной строки, `ku` и `tx`. Чтобы установить `python` 0.42 для Python 3 в виртуальной среде (`venv`), используйте следующее:

```
$ python3 -m venv /tmp/pycoin
$ . /tmp/pycoin/bin/activate
$ pip install pycoin==0.42
Downloading/unpacking pycoin==0.42
  Downloading pycoin-0.42.tar.gz (66kB): 66kB downloaded
  Running setup.py (path:/tmp/pycoin/build/pycoin/setup.py) egg_info for package
pycoin

Installing collected packages: pycoin
  Running setup.py install for pycoin

    Installing tx script to /tmp/pycoin/bin
    Installing cache_tx script to /tmp/pycoin/bin
    Installing bu script to /tmp/pycoin/bin
    Installing fetch_unspent script to /tmp/pycoin/bin
    Installing block script to /tmp/pycoin/bin
    Installing spend script to /tmp/pycoin/bin
    Installing ku script to /tmp/pycoin/bin
    Installing genwallet script to /tmp/pycoin/bin
Successfully installed pycoin
Cleaning up...
$
```

Вот пример сценария на Python для загрузки и траты некоторого количества биткоинов с помощью библиотеки `pycoin`:

```
#!/usr/bin/env python

from pycoin.key import Key

from pycoin.key.validate import is_address_valid, is_wif_valid
from pycoin.services import spendables_for_address
from pycoin.tx.tx_utils import create_signed_tx

def get_address(which):
    while 1:
        print("enter the %s address=> " % which, end='')
        address = input()
        is_valid = is_address_valid(address)
        if is_valid:
            return address
        print("invalid address, please try again")

src_address = get_address("source")
spendables = spendables_for_address(src_address)
print(spendables)

while 1:
    print("enter the WIF for %s=> " % src_address, end='')
    wif = input()
    is_valid = is_wif_valid(wif)
    if is_valid:
        break
    print("invalid wif, please try again")

key = Key.from_text(wif)
if src_address not in (key.address(use_uncompressed=False),
key.address(use_uncompressed=True)):
    print("** WIF doesn't correspond to %s" % src_address)
print("The secret exponent is %d" % key.secret_exponent())

dst_address = get_address("destination")

tx = create_signed_tx(spendables, payables=[dst_address], wifs=[wif])

print("here is the signed output transaction")
print(tx.as_hex())
```

Для примера использования утилит командной строки ku и tx см [\[appdxbitcoinimpproposals\]](#).

btcd

btcd — это реализация полного узла Биткоин, написанная на языке Go. В настоящее время она полностью соответствует базовому клиенту bitcoind, включая повторение ошибок. Она также корректно ретранслирует вновь найденные блоки, поддерживает пул транзакций, и ретранслирует отдельные операции, которые еще не успели попасть в блок. Это гарантирует, что все отдельные сделки, принятые в пул будут следовать требуемым правилам, а также включает в себя подавляющее большинство более строгих проверок, требуемых майнерами ("стандартные" транзакции).

Одно из ключевых различий между btcd и bitcoind состоит в том, что btcd не включает в себя функцию бумажника и это было сделано намерено. Это означает, что вы не можете на самом деле создавать или получать платежи непосредственно с помощью btcd. Подобная функциональность обеспечивается проектами btcwallet и btcgui, оба находящихся в стадии активной разработки. Другие заметные различия между btcd и bitcoind включают поддержку в btcd как запросов HTTP POST, так и предпочтительных WebSocket, а также тем фактом, что соединения RPC со включенным TLS по умолчанию.

Установка btcd

Для того, чтобы установить btcd под Windows, загрузите и запустите msi-инсталлятор, доступный по ссылке [GitHub](#), или выполните следующую команду на Linux, если у вас уже установлен язык Go:

```
$ go get github.com/conformal/btcd/...
```

Чтобы обновить btcd до последней версии, просто выполните:

```
$ go get -u -v github.com/conformal/btcd/...
```

Управление btcd

У btcd есть несколько опций конфигурации, которые можно просмотреть, запустив:

```
$ btcd --help
```

btcd поставляется с утилитой командной строки btcctl, которая может быть использована как для управления, так и для запросов к btcd с помощью RPC. По умолчанию RPC-сервер в btcd не активирован; необходимо как минимум задать имя пользователя и пароль для RPC в следующих конфигурационных файлах:

- *btcd.conf*:

```
[Application Options]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

- *btctl.conf*:

```
[Application Options]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

Если вы хотите переопределить конфигурацию из командной строки:

```
$ bitcoind -u myuser -P SomeDecentp4ssw0rd
$ btctl -u myuser -P SomeDecentp4ssw0rd
```

Для того, чтобы получить список доступных параметров, запустите:

```
$ btctl --help
```

Ключи, адреса, кошельки

Введение

Владение биткоинами устанавливается через *цифровые ключи*, *Биткоин-адреса* и *цифровые подписи*. Цифровые ключи не хранятся в сети, а вместо этого создаются и сохраняются пользователями в файле, или простой базе данных, называемой *кошельком*. Цифровые ключи в кошельке пользователя полностью независимы от протокола Биткоин и создаются и управляются с помощью программного обеспечения кошелька пользователя без ссылки на блокчейн или доступа к сети Интернет. Благодаря ключам становятся возможными многие из интересных свойств Биткоин, в том числе децентрализованного доверия и контроля, подтверждения владения, и модель безопасности, основанная на криптографическом доказательстве.

Каждая транзакция в Биткоин требует включения в блокчейн валидной подписи, которая может быть сгенерирована только при помощи валидных цифровых ключей; Поэтому, любой, кто обладает копией этих ключей, имеет контроль над средствами в этой учетной записи. Ключи состоят из пары: приватного (секретного) и публичного ключей. Публичный ключ похож на номер банковского счета, а приватный ключ аналогичен PIN-коду или подписи на чеке, который позволяет доступ к учетной записи. Эти цифровые ключи очень редко видны пользователями Биткоин. По большей части, они хранятся в файле бумажника и управляются программным обеспечением кошелька.

В платежной части Биткоин-транзакции, открытый ключ адресата представлен в виде его цифрового отпечатка, который называется *Биткоин-адресом*, и который аналогичен имени получателя средств на банковском чеке. В большинстве случаев Биткоин-адрес генерируется из и соответствует публичному ключу. Тем не менее, не все Биткоин-адреса представляют публичные ключи; они также могут представлять других получателей средств, например сценарии, как мы увидим позже в этой главе. Таким образом, Биткоин-адреса абстрагируют получателя средств, будь то физическое или юридическое лицо. Биткоин-адрес является единственным представлением ключей, с которым пользователи будут регулярно сталкиваться, т.к. именно им придется поделиться с миром.

В этой главе мы познакомимся с кошельками, которые содержат криптографические ключи. Мы посмотрим как ключи генерируются, хранятся и управляются. Мы рассмотрим различные форматы, используемые для представления приватных и публичных ключей, адреса и адресов сценариев. Наконец, мы ознакомимся со специальным применением ключей: для подписи сообщений, для доказательства прав собственности, а также для создания "тщеславных" адресов и бумажных кошельков.

Криптография с открытым ключом и криптовалюта

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

С момента изобретения криптографии с открытым ключом, было открыто несколько подходящих математических функций, таких как возведение в степень простого числа и умножения эллиптических кривых. Эти математические функции практически необратимы, это означает, что результат этих функций легко получить в одном направлении и невозможно в обратном. На основании этих математических функций, криптография позволяет создание цифровых шифров и неподдельных цифровых подписей. В Биткоин используется умножение эллиптических кривых.

В Биткоин для создания пары ключей, контролирующих доступ к средствам, используется криптография с открытым ключом. Пара ключей состоит из приватного ключа и, производного от него, уникального публичного ключа. Публичный ключ используется для получения биткоинов, а закрытый ключ используется для подписи транзакций, способных потратить эти биткоины.

Между публичным и приватным ключами существует математическое соотношение, которое позволяет подписывать сообщения приватным ключом, а эту подпись затем можно проверить при помощи публичного ключа, не раскрывая при этом приватную часть.

Когда некто хочет потратить свои биткоины, он предоставляет свой открытый ключ и подпись (каждый раз разную, но созданную на основе все того же приватного ключа) в теле транзакции. Путем публикации публичного ключа и подписи, все узлы сети могут проверить и принять транзакцию действительной, подтверждая, что лицо, передающее права на средства, является их владельцем.

ТИП

В большинстве реализаций кошельков, приватные и публичные ключи хранятся вместе как *пара ключей* для удобства. Однако публичный ключ может быть вычислен из приватного ключа, так что возможно хранить только приватный ключ.

Приватные и публичные ключи

Биткоин-кошелек содержит коллекцию из пар ключей, каждая из которых состоит из приватного и публичного ключа. Приватный ключ (k) представляет собой число, как правило, взятое наугад. Используя приватный ключ, при помощи односторонней функции умножения эллиптических кривых, мы получаем публичный ключ (K). Из публичного ключа (K), используя одностороннюю функцию криптографического хэша мы получаем Биткоин-адреса (A). В этом разделе мы начнем с создания приватного ключа, взглянем на математику эллиптической кривой, которая используется для превращения его в публичный ключ и, наконец, создадим Биткоин-адрес из публичного ключа. Отношения между приватным ключом, публичным ключом и Биткоин-адресом показано в [Приватный ключ, публичный ключ, и Биткоин-адрес](#).

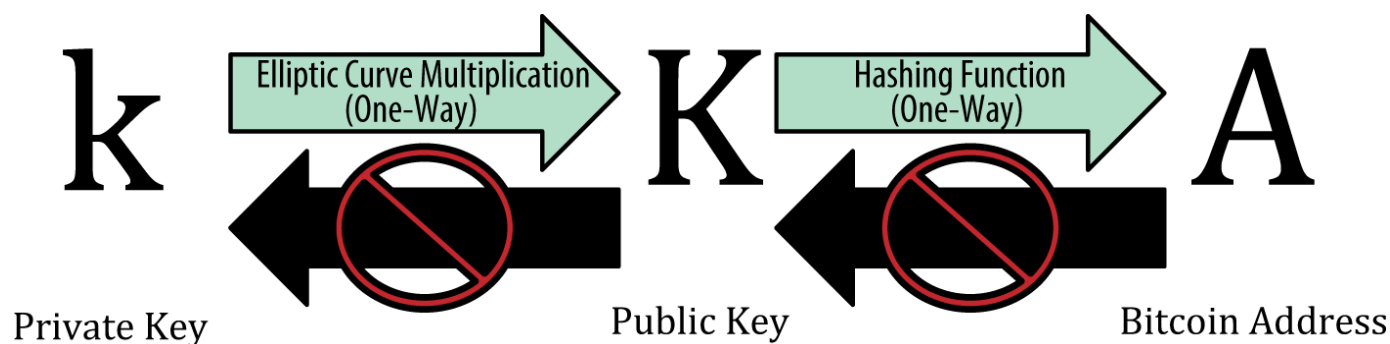


Figure 1. Приватный ключ, публичный ключ, и Биткоин-адрес

Приватные ключи

Приватный ключ — это просто число, взятое наугад. Владение приватным ключом необходимо для контроля пользователя над средствами, связанными с соответствующим Биткоин-адресом. Приватный ключ используется для создания подписи, которая необходима в качестве доказательства владения средствами в транзакции. Приватный ключ хранится в строжайшем секрете, потому что его утечка в другие руки эквивалентно передачи контроля над средствами, запертыми этим ключом. Приватный ключ также должен иметь резервную копию и защищен от случайной потери, потому что если он будет потерян, то и средства так же будут потеряны навсегда.

TIP

Приватный ключ — это просто число. Можете взять случайный приватный ключ, используя только монету, карандаш и бумага: бросить монету 256 раз, и у вас есть двоичное число случайного приватного ключа, который можно использовать в Биткоин-кошельке. Публичный ключ может быть сгенерирован из приватного ключа.

Создание приватного ключа из случайного числа

Первый и самый важный шаг при генерации ключей — это найти надежный источник энтропии, или случайности. Создание ключа Bitcoin, по существу, то же самое, что "взять число между 1 и 2^{256} ." Точный способ выбора числа не имеет значения до тех пор, пока не предсказуем или повторим. Программное обеспечение Биткоин использует генераторы случайных чисел операционной системы. Как правило, генератор случайных чисел ОС инициализируется источником случайности пользователя, поэтому вам может быть предложено случайно пошевелить мышью в течение нескольких секунд. Для настоящих параноиков, ничто не сравнится с костями, карандашом, и бумагой.

Более точно, приватный ключ может быть любым числом между 1 и $n - 1$, где n константа ($n = 1.158 * 10^{77}$, немного меньше, чем 2^{256}) определяется как порядок эллиптической кривой используемой в Bitcoin (см. [Криптография эллиптических кривых](#)). Чтобы создать подобный ключ, мы случайно выбираем 256-битное число, и убеждаемся, что оно меньше, чем $n - 1$. В терминах программирования, это обычно достигается путем подачи большой строки случайных битов, взятых из криптографически стойкого источника случайности, на вход хэш-алгоритма SHA256, который выдаст удобное 256-битное число. Если результат меньше, чем $n -$

1, мы получаем подходящий приватный ключ. В противном случае, мы просто пробуем еще раз с другим случайным числом.

ТИП Не пишите свой собственный генератор случайных чисел и не используйте генератор по умолчанию, предоставляемый вашим языком программирования. Используйте криптографически стойкий генератор псевдослучайных чисел (ГПСЧ) с зерном из источника достаточной энтропии. Изучите документацию выбранной вами библиотеки генератора случайных чисел и убедитесь, что она криптографически безопасна. Правильная имплементация и употребление ГПСЧ имеет критическое значение для безопасности ключей.

Ниже приведен случайным образом сгенерированный секретный ключ (k) в шестнадцатеричном формате (256 двоичных цифр в виде 64 шестнадцатеричных цифр, каждая по 4 бита):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

ТИП Размер пространства возможных приватных ключей Биткоин, 2^{256} — это неизмеримо большое количество. Это примерно 10^{77} в десятичной. Видимая Вселенная, по оценкам, имеет 10^{80} атомов.

Для генерации нового ключа клиентом Bitcoin Core (см [\[ch03_bitcoin_client\]](#)), используйте команду `getnewaddress`. По соображениям безопасности она отображает только публичный ключ, но не приватный. Для того, чтобы заставить `bitcoind` показать приватный ключ, используйте команду `dumpprivkey`. Команда `dumpprivkey` показывает приватный ключ в формате Base58 с контрольной суммой, который называется *Wallet Import Format* (WIF), и который мы рассмотрим более подробно в [Форматы приватных ключей](#). Вот пример генерирования и отображения приватного ключа, используя две вышеупомянутые команды:

```
$ bitcoind getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
$ bitcoind dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Команда `dumpprivkey` открывает файл кошелька и извлекает приватный ключ, который был создан при помощи команды `getnewaddress`. `Bitcoind` не может узнать приватный ключ из публичного, иначе если они оба не хранятся в кошельке.

ТИП Команда `dumpprivkey` не генерирует приватный ключ из открытого ключа, поскольку это невозможно. Команда просто показывает приватный ключ, который уже известен кошельку и который был создан с помощью команды `getnewaddress`.

Также можно использовать утилиту командной строки Bitcoin Explorer (см [\[libbitcoin\]](#)) для

генерации и просмотра приватных ключей при помощи команд `seed`, `ec-new` и `ec-to-wif`:

```
$ bx seed | bx ec-new | bx ec-to-wif  
5J3mBbAN58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Публичные ключи

Публичный ключ вычисляется из приватного ключа с помощью необратимого умножения на эллиптических кривых: $(K = k * G)$, где k — это приватный ключ, G — константная точка, называемая *генераторной точкой*, а K — результирующий публичный ключ. Обратная операция, известная как "нахождение дискретного логарифма", вычисление k при известном K возможна только при помощи полного перебора k , т.е. лобовой атаки. Прежде, чем мы продемонстрируем, как создать публичный ключ из приватного, давайте взглянем на криптографию на эллиптических кривых немного более подробно.

Криптография эллиптических кривых

Криптография на эллиптических кривых — это вид асимметричной криптографии или криптографии с открытым ключом, основанной на проблеме дискретного логарифмирования на эллиптических кривых.

На [Эллиптическая кривая](#) изображен пример эллиптической кривой, аналогичной той, что используется в Биткоин.

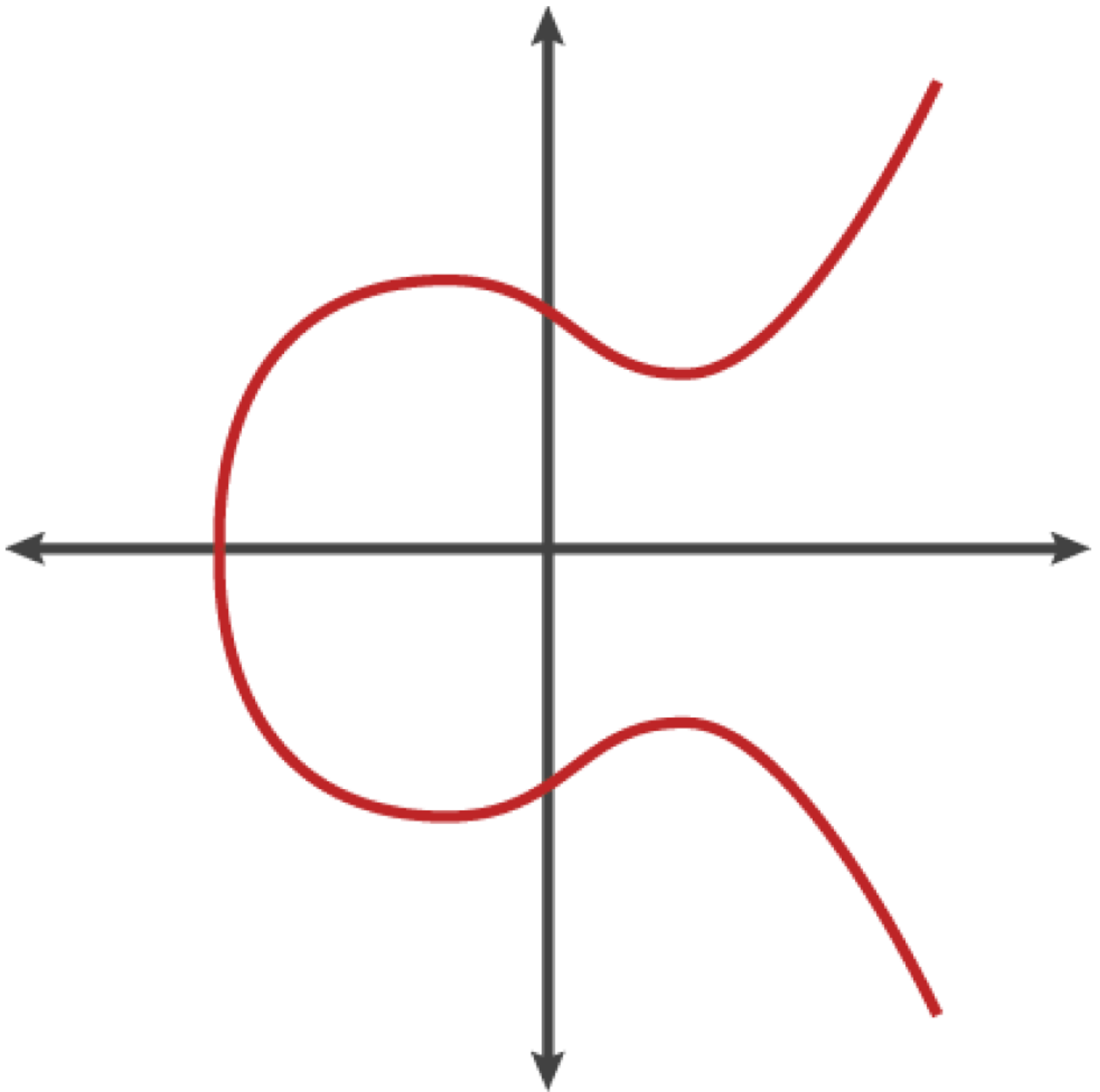


Figure 2. Эллиптическая кривая

В Биткоин используется определенная эллиптическая кривая и набор математических констант из стандарта под названием `secp256k1` установленного Национальным Институтом Стандартов и Технологий (NIST). Кривая `secp256k1` определяется следующей функцией эллиптической кривой:

или

$\text{mod } p$ (модуль простого числа p) показывает, что эта кривая над конечным полем простого порядка p , также может быть записана как $\mathbb{F}_p$, где $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ — это очень большое простое число.

Так как эта кривая определена над конечным полем простого порядка, а не вещественных чисел, она выглядит как узор из точек, рассеянных в двух измерениях, что трудно визуализировать. Тем не менее, математика идентична математике эллиптической кривой в вещественных числах. В качестве примера, на [Криптография эллиптических кривых: визуализация эллиптической кривой над \$F\(p\)\$, при \$p=17\$](#) изображена та же эллиптическая кривая над гораздо меньшего конечного поля простого порядка 17, показывающий структуру точек на координатной сетке. Эллиптическая кривая Биткоин, `secp256k1` может рассматриваться как гораздо более сложная картина точек на неизмеримо большой координатной сетке.

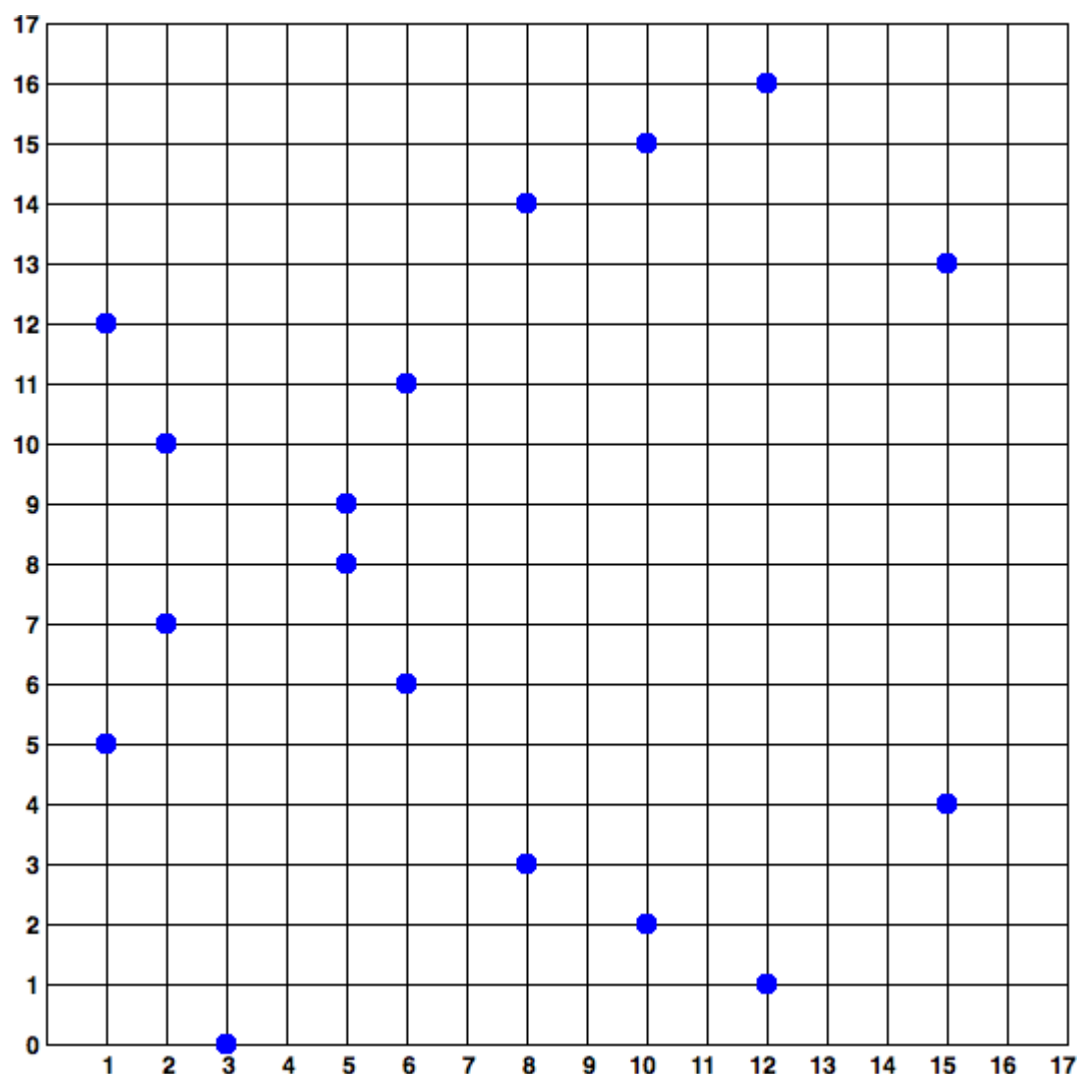


Figure 3. Криптография эллиптических кривых: визуализация эллиптической кривой над $F(p)$, при $p=17$

Так, например, точка P с координатами (x, y) , что является точкой на кривой `secp256k1`. Это можно проверить самостоятельно, используя Python:

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

```

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 точка, называемая "точкой на бесконечности", что примерно соответствует роли 0 в сложении. На компьютерах, это иногда представлена как $x = y = 0$ (что не удовлетворяет уравнению эллиптической кривой, но это простой отдельный случай, который может быть установлен).

Есть также оператор "сложения", имеющий некоторые свойства аналогичные традиционному сложению действительных чисел, знакомому еще со школы. Имея две точки P_1 и P_2 на эллиптической кривой, существует и третья точка такая, что $P_3 = P_1 + P_2$.

Геометрически эта третья точка P_3 может быть найдена путем прочерчивания линии между P_1 и P_2 . Эта линия пересечет эллиптическую кривую ровно в одной дополнительной точке. Назовем эту точку $P_3' = (x, y)$. А затем найдем точку P_3 с координатами $(x, -y)$.

Есть несколько особых случаев, которые объясняют необходимость "точки в бесконечности."

Если P_1 и P_2 одна и та же точка, то прямая, проведенная "между" P_1 и P_2 должна идти по касательной к кривой в этой точке P_1 . Эта касательная пересечет кривую ровно в одной новой точке. Вы можете использовать методы из тригонометрии для определения угла наклона касательной. Любопытно, но эти методы работают даже если мы ограничиваемся точками на кривой с целочисленными координатами!

В некоторых случаях (например, если P_1 и P_2 имеют одинаковые значения x -координаты, но разные значения y), касательная пройдет строго вертикально, в этом случае $P_3 =$ "точка в бесконечности".

Если P_1 является "точкой на бесконечности", то сумма $P_1 + P_2 = P_2$. Аналогично, если P_2 является точкой на бесконечности, то $P_1 + P_2 = P_1$. Это показывает, как точка на бесконечности играет роль 0.

Оказывается, что $+$ ассоциативно, что означает, что $(A + B) + C = A + (B + C)$. Это означает, что мы можем написать $A + B + C$ без скобок и без какой-либо двусмысленности.

Теперь, когда мы определили сложение, мы можем определить умножение в смысле, расширяющем сложение. Для точки P на эллиптической кривой, если k является целым

числом, то $kP = P + P + P + \dots + P$ (k раз). Обратите внимание, что k иногда сбивающе с толку называется "экспонентой".

Создание публичного ключа

Начиная с закрытого ключа в виде случайного числа k , мы умножаем его на заданную точку кривой называемую *точкой генерации* G для того, чтобы получить еще какую-то точку кривой, которая и будет являться соответствующим открытым ключом K . Точка генерации определена как часть стандарта `secp256k1` и всегда одна и та же для ключей в Биткойн:

где k — это приватный ключ, G — генераторная точка, а K — это результирующий публичный ключ, точка на кривой. Так как генераторная точка всегда одинакова для всех пользователей Биткойн, приватный ключ k при умножении на G всегда даст один и тот же публичный ключ K . Отношение между k и K зафиксировано, но может быть вычислено только в одном направлении, от k к K . Вот почему Биткойн-адресом (полученным из K) можно делиться с кем угодно без риска раскрыть приватный ключ (k).

ТИП

Приватный ключ может быть преобразован в открытый ключ, а публичный ключ не может быть преобразован обратно в приватный ключ, так как математически возможно только одностороннее преобразование.

Реализуя умножение на эллиптической кривой, мы берем приватный ключ k сгенерированный ранее и умножаем его на генераторную точку G для нахождения публичного ключа K :

```
K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G
```

Публичный ключ K определен как точка $K = (x, y)$:

$K = (x, y)$

где,

$x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A$

$y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB$

Для визуализации умножения точки на целое число, мы будем использовать простейшую эллиптическую кривую в вещественных числах, потому что математика та же самая. Наша цель, чтобы найти несколько kG генераторной точки G . Это то же самое, что суммирование G само с собой k раз подряд. В эллиптических кривых, сложение точки самой с собой является эквивалентом построения касательной в точке и нахождения пересечения с кривой в другой точке, а затем проекции этой точки на оси абсцисс.

На рисунке [Криптография эллиптических кривых: Визуализация умножения точки \$G\$ на целое](#)

k на эллиптической кривой. виден процесс получения точек G , $2G$, $4G$ при помощи геометрических операций на кривой.

TIP

Большинство реализаций Биткоин для работы с эллиптическими кривыми используют [криптографическую библиотеку OpenSSL](#).. Например, для получения публичного ключа используется функция `EC_POINT_mul()`.

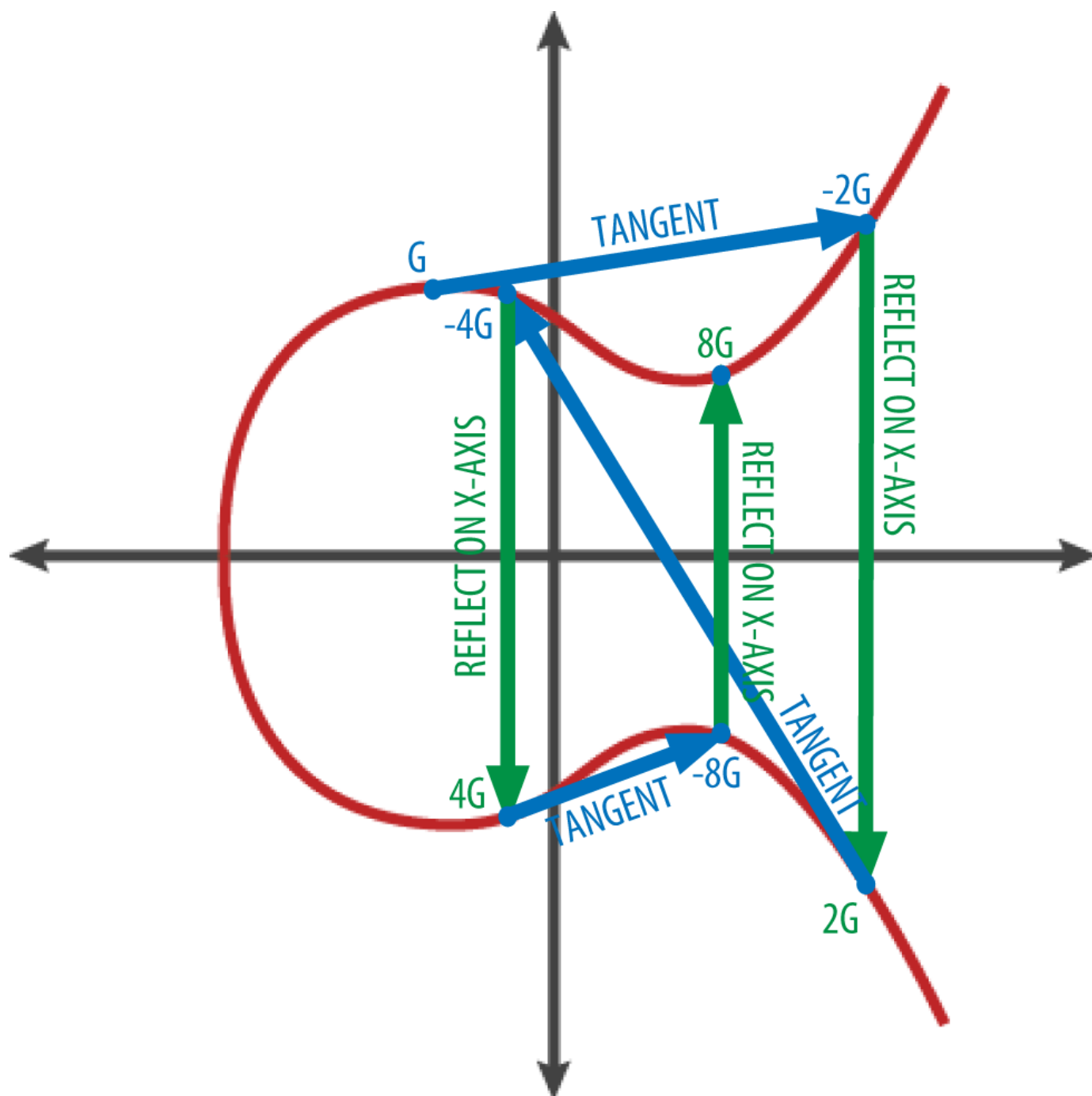


Figure 4. Криптография эллиптических кривых: Визуализация умножения точки G на целое k на эллиптической кривой.

Биткоин-адреса

Биткоин-адрес представляет собой строку из цифр и символов латинского алфавита. Адреса, полученные из публичных ключей начинаются с цифры "1". Вот пример адреса:

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

Чаще всего в качестве получателя средств в транзакциях Биткоин фигурирует именно Биткоин-адрес. Если сравнить Биткоин-транзакцию с бумажным чеком, то адрес получателя явился бы тем, что мы пишем в строке после "Укажите получателя платежа." Получателем на бумажном чеке иногда может выступать имя владельца банковского счета, но может быть и название фирмы, учреждения, или даже приказ обналичить. Так как на бумажных чеках можно не указывать номер счета, а использовать абстрактное имя в качестве получателя средств, то эти чеки становятся очень гибким платежным инструментом. Биткоин-транзакции используют похожую абстракцию — адрес в сети Биткоин. Биткоин-адрес может представлять владельца пары, состоящей из приватного/публичного ключа, или может представлять кое-что еще, например, платежный сценарий, которые мы рассмотрим в [\[p2sh\]](#). Пока что давайте рассмотрим простой случай, Биткоин-адрес, представляющий и производный от публичного ключа.

Биткоин-адрес получается из публичного ключа путем использования односторонней криптографической операции хеширования. А "алгоритм хеширования" или просто "алгоритм хеша" — это односторонняя функция, производящая отпечаток или "хэш" произвольного количества данных на входе. Криптографические хэш-функции широко используются в Биткоин: в Биткоин-адресах, в адресах сценариев, и в алгоритме доказательства работы в мейннете. Адрес из публичного ключа получается в результате применения алгоритма под названием Secure Hash Algorithm (SHA) и RACE Integrity Primitives Evaluation Message Digest (RIPEMD), конкретно SHA256 and RIPEMD160.

Начиная с открытого ключа K , мы вычисляем SHA256-хэш и результат пропускаем через хэш-алгоритм RIPEMD160, получив на выходе 160-ти битное (20-байт) число:

где K — это публичный ключ и A результирующий Биткоин-адрес.

ТИП

Биткоин-адрес — это *не* то же самое, что публичный ключ. Биткоин-адреса получаются из публичного ключа при помощи односторонней функции.

Биткоин-адреса почти всегда представлены пользователям в кодировке называемой "Base58Check" (см. [Кодировки Base58 и Base58Check](#)), в которой используется 58 символов (формат Base58) и контрольная сумма для повышения читаемости, избежания двусмысленностей в написании, и защиты от ошибок. Base58Check используется и в во многих других случаях, когда важно корректное прочтение или транскрибирование числа, Биткоин-адреса, приватного ключа, зашифрованного ключа, или хеша сценария. В следующем разделе мы рассмотрим механику кодирования и декодирования Base58Check, и результирующие

представления. В [Публичный ключ в Биткоин-адрес: преобразование публичного ключа в адрес Биткоин](#) иллюстрируется превращение публичного ключа в Биткоин-адрес.

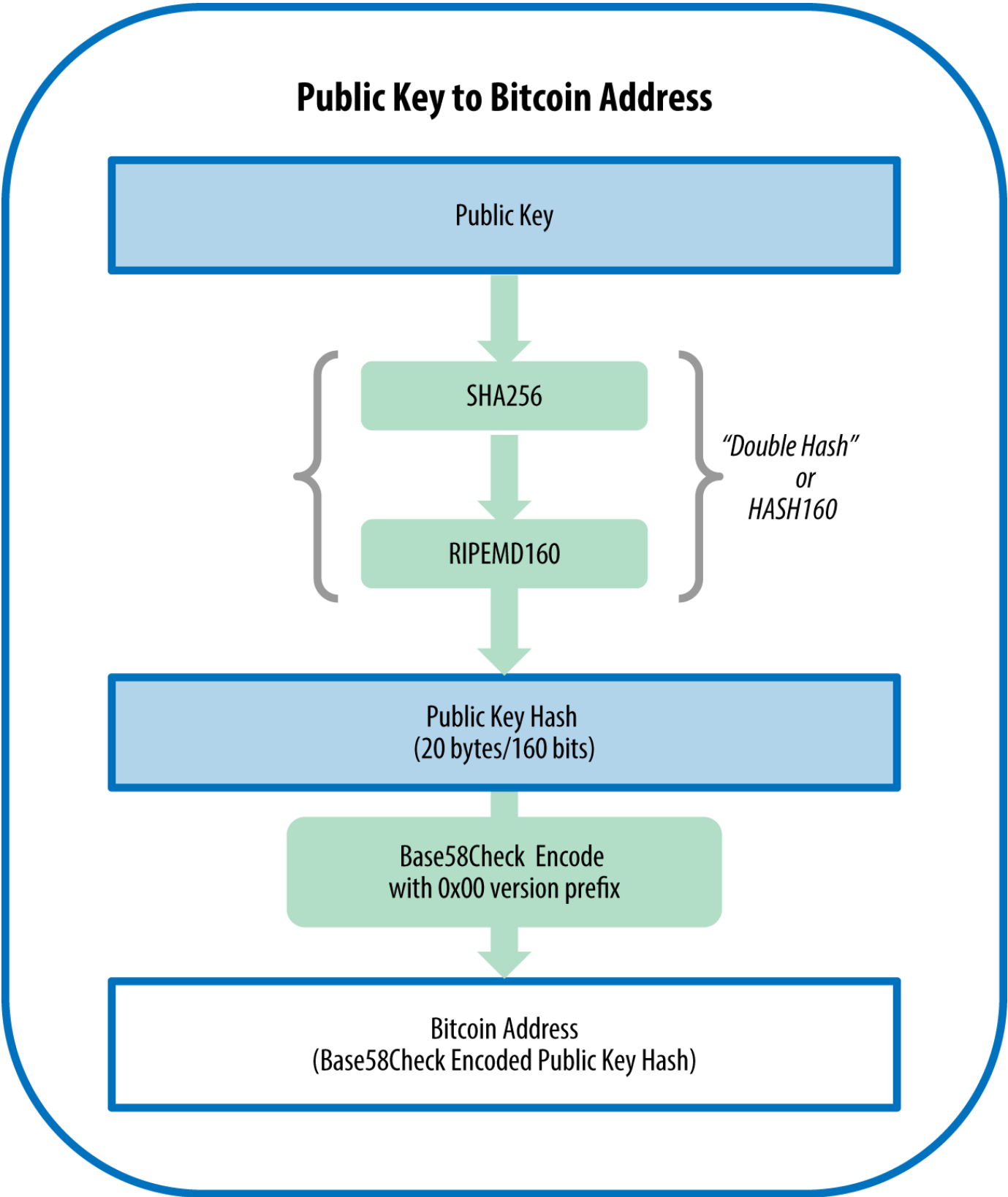


Figure 5. Публичный ключ в Биткоин-адрес: преобразование публичного ключа в адрес Биткоин

Кодировки Base58 и Base58Check

Для того, чтобы представлять длинные числа в компактном виде, используя меньше символов, многие компьютерные системы используют смешанные буквенно-цифровые представления с базой (или систем счисления) выше, чем 10. Например, в то время как традиционная десятичная система использует 10 цифр от 0 до 9, шестнадцатеричная система использует 16, с буквами от A до F в качестве шести дополнительных символов. Число, представленное в шестнадцатеричном формате короче, чем эквивалентное в десятичном представлении. Еще более компактное, представление Base-64 используется для передачи бинарных данных в "текстовых средах", например системы email. Алфавит Base-64 состоит из 26 прописных букв, 26 заглавных букв, 10 цифр, и еще двух символов: "+" и "/". Base-64 чаще всего используется для кодирования вложений в сообщениях электронной почты. Формат кодирования Base58 разработан для использования в Биткоин и используется во многих других криптовалютах. Он предлагает баланс между компактным представлением, читаемостью, определением и предотвращением ошибок. Base58 — это подмножество Base64, использующее прописные и заглавные буквы и цифры, но без некоторых символов, которые часто ошибочно принимают за друг друга и могут отображаться идентично некоторыми шрифтами. В частности, Base58 — это Base64 без числа 0 (ноль), O (заглавная буква O), l (маленькая L), I (большая i), и символов "\", " и "/". Или, проще говоря, это набор прописных и заглавных букв и цифр без четырех (0, O, L, I), упомянутых выше.

Example 1. алфавит Base58

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

Чтобы добавить дополнительную защиту от опечаток или ошибок транскрипции, Base58Check — это Base58 со встроенным кодом проверки ошибок. Четыре байта контрольной суммы добавлены в конец кодируемых данных. Контрольная сумма получена хэшированием закодированных данных и, следовательно, может быть использована для обнаружения и предотвращения ошибок транскрипции и опечаток. Программное обеспечение вычислит контрольную сумму данных при декодировании и сравнит его с контрольной суммой из кода. Несовпадение будет означать внесенную ошибку, а данные Base58Check недействительными. Например, подобная проверка предотвращает возможность послать средства по несуществующему Биткоин-адресу и таким образом потере средств.

Для конвертирования данных (числовых) в формат Base58Check, к данным сначала требуется добавить префикс, называемый "байтом версии", который служит для определения типа кодируемых данных. Например, в случае Биткоин-адреса префикс 0 (0x00 в шестнадцатеричной системе), в то время как префикс для приватного ключа — это 128 (0x80 в шестнадцатеричной системе). Список общих префиксов версий приведен в [Префикс версии в Base58Check и примеры закодированного результата](#).

Далее мы вычисляем контрольную сумму "удвоенный SHA", в смысле, что применяем хеш-

алгоритм SHA256 дважды на предыдущем результате (префиксе и данных).

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

Из полученного 32-ухбайтного хэша (хэш хэша), мы берем только первые четыре байта. Эти четыре байта служат в качестве кода проверки ошибочности, или контрольной суммы. Эта контрольная сумма затем добавляется к концу строки.

Результат состоит из трех элементов: префикса, данных и контрольной суммы. Этот результат кодируется с использованием описанного раньше алфавита Base58. В [Кодировка Base58Check: кодировка Base58 с добавлением версии, контрольной суммы для однозначного кодирования](#) иллюстрируется процесс кодирования в формат Base58Check.

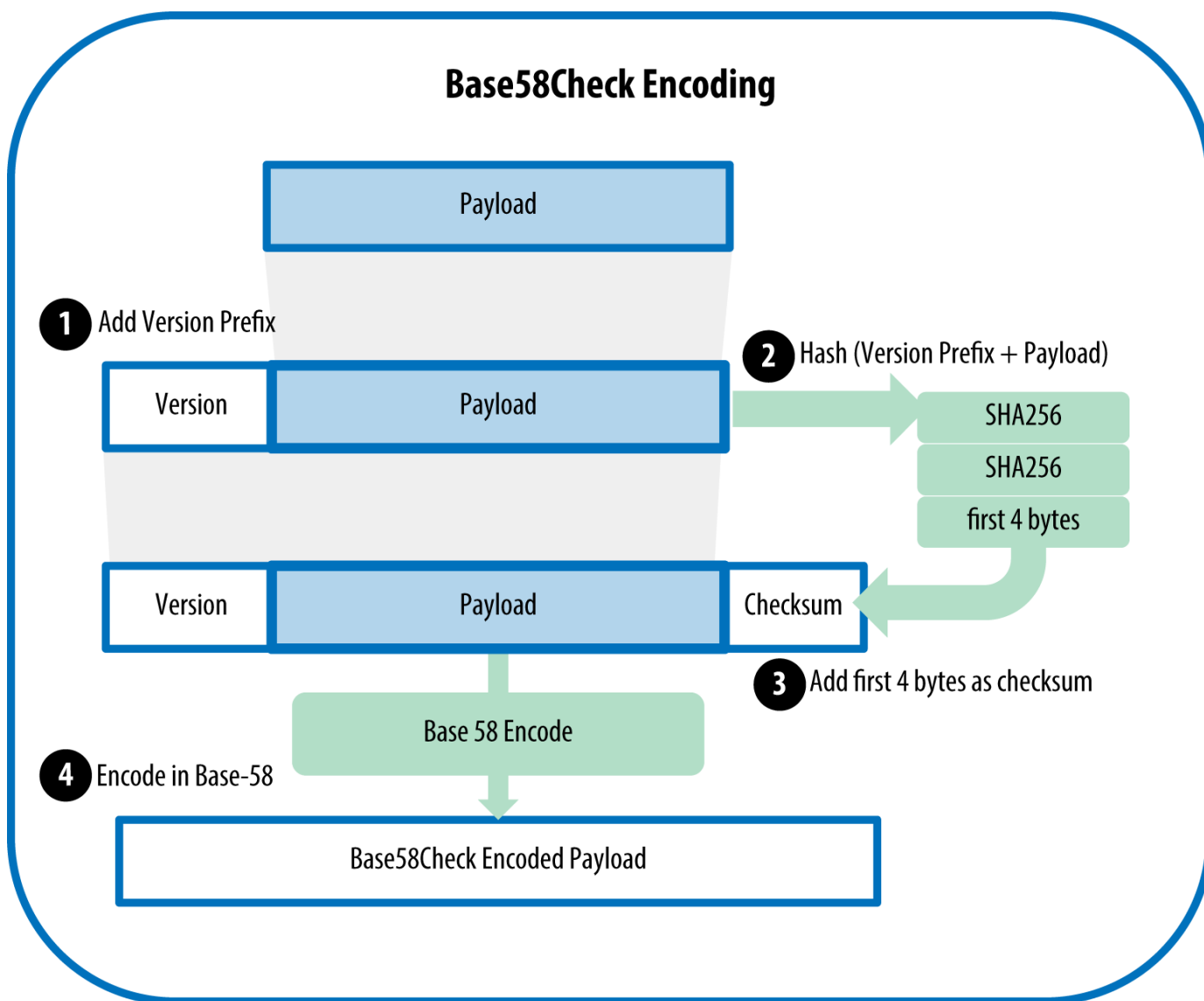


Figure 6. Кодировка Base58Check: кодировка Base58 с добавлением версии, контрольной суммы для однозначного кодирования

В Биткоин, большинство данных представлены пользователем в кодировке Base58Check,

потому что это представление компактно, удобно для чтения, и помогает легко обнаружить ошибки. Префикс версии в кодировке Base58Check используется для создания различных форматов, легко отличаемых по определенным символам в начале строк. Эти символы позволяют людям легко определять закодированный тип данных. Например, Биткоин-адрес в кодировке Base58Check начинается с 1, а приватный ключ формата WIF в кодировке Base58Check начинается с 5. Примеры префиксов и результирующих символов алфавита Base58 показаны в [Префикс версии в Base58Check и примеры закодированного результата](#).

Table 1. Префикс версии в Base58Check и примеры закодированного результата

Type	Version prefix (hex)	Base58 result prefix
Биткоин-адрес	0x00	1
Pay-to-Script-Hash адрес	0x05	3
Testnet-адрес	0x6F	m или n
Приватный ключ WIF	0x80	5, K или L
ВР38 зашифрованный приватный ключ	0x0142	6P
ВР32 Расширенный публичный ключ	0x0488B21E	xpub

Давайте посмотрим на весь процесс создания Биткоин-адреса, от приватного ключа, к публичному ключу (точке на эллиптической кривой), адресу после двойного хеширования и, наконец, после кодирования в Base58Check. C++ код в [Создаем Биткоин-адрес в кодировке Base58Check из приватного ключа](#) показывает полный процесс шаг-за-шагом. Пример кода использует библиотеку libbitcoin, описанную в [\[alt_libraries\]](#).

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

int main()
{
    // Private secret key.
    bc::ec_secret secret;
    bool success = bc::decode_base16(secret,
        "038109007313a5807b2eccc082c8c3fbb988a973cacf1a7df9ce725c31b14776");
    assert(success);
    // Get public key.
    bc::ec_point public_key = bc::secret_to_public_key(secret);
    std::cout << "Public key: " << bc::encode_hex(public_key) << std::endl;

    // Create Bitcoin address.
    // Normally you can use:
    //   bc::payment_address payaddr;
    //   bc::set_public_key(payaddr, public_key);
    //   const std::string address = payaddr.encoded();

    // Compute hash of public key for P2PKH address.
    const bc::short_hash hash = bc::bitcoin_short_hash(public_key);

    bc::data_chunk unencoded_address;
    // Reserve 25 bytes
    //   [ version:1 ]
    //   [ hash:20   ]
    //   [ checksum:4 ]
    unencoded_address.reserve(25);
    // Version byte, 0 is normal BTC address (P2PKH).
    unencoded_address.push_back(0);
    // Hash data
    bc::extend_data(unencoded_address, hash);
    // Checksum is computed by hashing data, and adding 4 bytes from hash.
    bc::append_checksum(unencoded_address);
    // Finally we must encode the result in Bitcoin's base58 encoding
    assert(unencoded_address.size() == 25);
    const std::string address = bc::encode_base58(unencoded_address);

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

Код использует predetermined приватный ключ, поэтому, он выдает один и тот же

Биткоин-адреса при каждом запуске, как показано на [Компиляция и запуск addr](#).

Example 3. Компиляция и запуск addr

```
# Компиляция файла addr.cpp
$ g++ -o addr addr.cpp $(pkg-config --cflags --libs libbitcoin)
# Запуск исполняемого файла addr
$ ./addr
Public key: 0202a406624211f2abbd6c68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6EhcdU1fpEdX7913CK
```

Форматы ключей

Приватные и публичные ключи могут быть представлены в различных форматах. Все эти представления кодируют одно и то же число, даже если результат выглядит по-разному. Эти форматы используются, в основном для того, чтобы люди могли воспринимать на слух и произносить текст ключей без внесения ошибок.

Форматы приватных ключей

Приватный ключ может быть представлен в различных форматах, каждый из которых соответствует тому же самому 256-битному числу. В таблице [Представления приватных ключей \(форматы кодирования\)](#) показаны три распространенных формата, используемых для представления закрытых ключей.

Table 2. Представления приватных ключей (форматы кодирования)

Тип	Префикс	Описание
Шестнадцатичный	Нет	64 шестнадцатичных цифры
WIF	5	Кодировка Base58Check: Base58 с префиксом версии 128 и 32-битной контрольной суммой
Сжатый WIF	K или L	То же, что выше, но с добавлением суффикса 0x01 перед кодированием

В таблице [Пример: один и тот же ключ в разных форматах](#) показан приватный ключ в этих трех форматах.

Table 3. Пример: один и тот же ключ в разных форматах

Формат	Приватный ключ
Шестнадцатиричный	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbkneyhfsYB1Jcn
Сжатый WIF	KxFC1jmwWCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Все эти представления просто различные способы закодировать одно и то же число, один и тот же приватный ключ. Они выглядят по-разному, но любой формат может быть легко преобразован в любой другой формат.

Мы используем команду wif-to-ec из Bitcoin Explorer (см. [libbitcoin](#)), чтобы показать, что оба WIF ключа представляют один и тот же приватный ключ:

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbkneyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec KxFC1jmwWCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

Декодирование из Base58Check

Команды Bitcoin Explorer (см. [libbitcoin](#)) позволяют легко писать сценарии оболочки и однострочники для управления Биткоин-ключами, адресами и транзакциями. Вы можете использовать Bitcoin Explorer для расшифровки формата Base58Check из командной строки.

Команда base58check-decode используется для декодирования несжатого ключа:

```
$ bx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbkneyhfsYB1Jcn
wrapper
{
  checksum 4286807748
  payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
  version 128
}
```

Результат содержит ключ, префикс версии Wallet Import Format (WIF) равный числу 128, и контрольную сумму.

Обратите внимание, что содержимое сжатого ключа добавляется с суффиксом 01, означающим, что производный публичный ключ должен быть сжат.

```
$ bx base58check-decode KxFC1jmwWCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
wrapper
{
  checksum 2339607926
  payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
  version 128
}
```

Перекодирование из шестнадцатеричного представления в Base58Check

Для того, чтобы закодировать в формат Base58Check (в противоположность предыдущей команде), мы используем команду `base58check-encode` из Bitcoin Explorer (см. [libbitcoin](#)), которой передадим приватный ключ в шестнадцатеричном виде с префиксом версии 128 для Wallet Import Format (WIF):

```
bx base58check-encode 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
--version 128
5J3mBbAN58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Перекодирование из шестнадцатеричного вида (сжатый ключ) в Base58Check

Для кодирования "сжатого" закрытого ключа в Base58Check (см. [Сжатые приватные ключи](#)), мы добавляем суффикс 01 к шестнадцатеричному представлению ключа и затем кодируем, как описано выше:

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version 128
KxFC1jmwWCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

В результате WIF-сжатый формат начинается с "K". Это означает, что приватный ключ в нем имеет суффикс "01" и будет использоваться для создания исключительно сжатых публичных ключей (см. [Сжатые публичные ключи](#)).

Форматы публичных ключей

Публичные ключи также представлены по-разному, но главное, в *сжатом* или *несжатом* видах.

Как мы видели ранее, публичный ключ — это точка на эллиптической кривой, имеющей координаты (x,y). Как правило он представлен префиксом 04 и двумя 256-битными числами после префикса : в первом числе x координата точки, а во втором y координата. Префикс 04 используется для того, чтобы можно было отличить несжатые публичные ключи от сжатых, которые начинаются с 02 или 03.

Вот открытый ключ, сгенерированный из приватного ключа, созданного нами ранее, в виде координат x and y :

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

Вот тот же самый публичный ключ показан в виде 520-битного числа (130 шестнадцатеричных цифр) с префиксом 04 и далее x и y в виде 04 x y :

```
K = 04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A<?pdf-
cr?>07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Сжатые публичные ключи

<?dbhtml orphans="4"?>Сжатые публичные ключи были введены для того, чтобы уменьшить размер транзакций и сэкономить место в блокчейне. Большинство транзакций включает публичный ключ, необходимый для проверки учетных данных владельца при платеже. Каждый публичный ключ требует 520 битов (префикс x y), которые, если их умножить на несколько сотен транзакций в блоке, или десятки тысяч транзакций в день, добавляют значительное количество к объему блокчейна.

Как мы видели в разделе [Публичные ключи](#), публичный ключ — это точка (x, y) на эллиптической кривой. Поскольку кривая выражает математическую функцию, точка на кривой представляет собой решение уравнения, а значит, если мы знаем x координату, то мы можем вычислить y координату, решив уравнение $y^2 \bmod p = (x^3 + 7) \bmod p$. Это позволяет хранить только x координату публичного ключа, опуская y координату и уменьшив размера ключа и пространство, необходимое для его хранения на 256 бит. Почти 50% уменьшение размера каждой транзакции, помогает сэкономить много пространства в течение долгого времени!

В то время как несжатые публичные ключи имеют префикс 04, сжатые начинаются либо с 02, либо с 03. Давайте разберемся почему возможных префиксов два: так как в левой части уравнения находится y^2 , это означает, что y может быть как положительным, так и отрицательным. На графике это означает, что в результирующая координата y может быть над осью x или под. Как можно видеть из графика эллиптической кривой в [Эллиптическая кривая](#), кривая симметрична относительно оси x . Так что, хоть мы и можем опустить значение координаты y , ее *знак* нам сохранить придется, или, другими словами, мы должны помнить, был ли он выше или ниже оси абсцисс, так как каждый из этих вариантов представляет собой различные точки и различные публичные ключи. При расчете эллиптической кривой в двоичной арифметике на конечной области простого порядка p , y координата имеет либо четное, либо нечетное значение, что соответствует знаку, как описано выше. Таким образом, для различения этих двух возможных значений y , мы храним сжатый публичный ключ с префиксом 02, если y четное, и, 03, если нечетно, позволяя программному обеспечению правильно выводиться координату y из x и распаковывать публичный ключ к полным

координатам точки. Сжатие публичного ключа показано в [Сжатие публичного ключа](#).

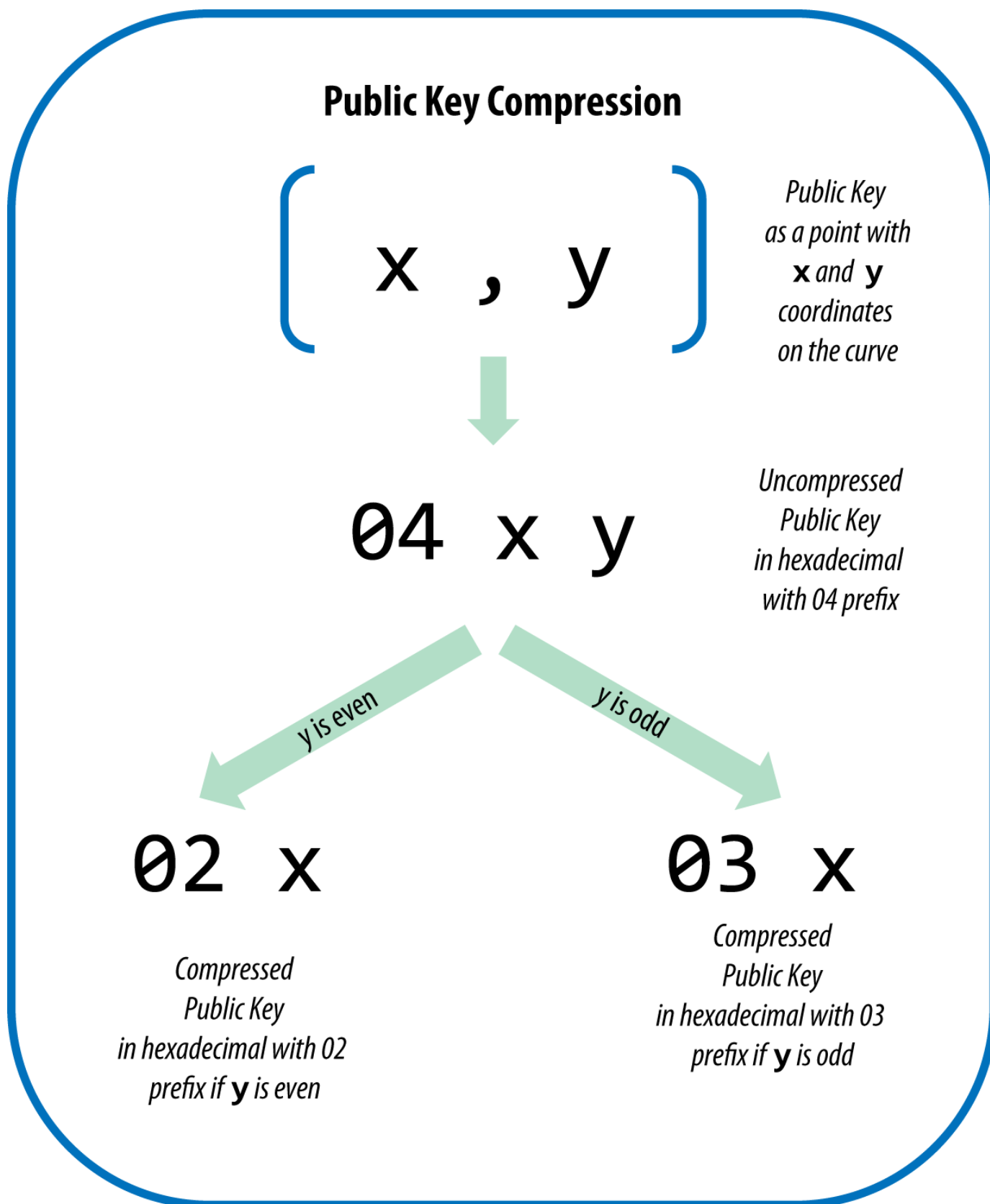


Figure 7. Сжатие публичного ключа

Вот тот же публичный ключ, сгенерированный ранее, показан как сжатый публичный ключ

длиной 264 бит (66 шестнадцатеричных цифр) с префиксом 03 указанием избыточной у координаты:

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Это сжатый публичный ключ соответствует тому же приватному ключу, что означает, что он был получен от того же приватного ключа. Тем не менее, он выглядит отличным от несжатого публичного ключа. Что еще более важно, если мы сконвертируем этот сжатый публичный ключ в Биткоин-адрес с помощью функции двойного хэширования (RIPEMD160(SHA256(K))), то будет получен *другой* Биткоин-адрес. Это может привести к путанице, так как означает, что от одного и того же приватного ключа может быть получен публичный ключ, выраженный в двух различных форматах (сжатом и несжатом), которым соответствуют два разных адреса Биткоин. Тем не менее, приватный ключ одинаков для обоих адресов.

Несжатые публичные ключи постепенно вытесняются в клиентском ПО, сжатые становятся по умолчанию, что оказывает значительное влияние на уменьшение размера транзакций и, следовательно, блокчейна. Тем не менее, не все клиенты пока поддерживают сжатые публичные ключи. Новые клиенты, поддерживающие сжатые публичные ключи должны поддерживать транзакции, созданные более старыми клиентами, не поддерживающими сжатые публичные ключи. Это особенно важно, когда приложение кошелька импортирует приватные ключи из другого приложения, потому что новый кошелек должен просканировать блокчейн в поиске транзакций, соответствующих этим импортируемым ключам. В поиске каких же Биткоин-адресов Биткоин-кошелек должен осуществить сканирование? Биткоин-адреса, полученные от несжатых или сжатых публичных ключей? В обоих случаях получаются валидные Биткоин-адреса, но они разные!

Чтобы решить эту проблему, когда приватные ключи экспортируются из кошелька, Wallet Import Format, используемый для их представления реализуется по-другому в новых Биткоин-кошельках, чтобы указать, что эти приватные ключи были использованы для получения *сжатых* публичных ключей и, следовательно, *сжатых* Биткоин-адресов. Это позволяет импортирующему кошельку отличить приватные ключи, происходящих от старых или новых кошельков и найти в блокчейне транзакции с адресами от несжатых или сжатых публичных ключей соответственно. В следующем разделе мы посмотрим как это работает более подробно.

Сжатые приватные ключи

Как ни странно, термин "сжатый приватный ключ" вводит в заблуждение, потому что, когда приватный ключ экспортируется, как WIF-сжатый это на самом деле добавляет один байт по сравнению с "несжатым" закрытым ключом. Это потому, что он имеет дополнительный суфикс 01, который подчеркивает, что он происходит от нового кошелька и должен быть использован только для получения сжатых публичных ключей. Приватные ключи не сжимаются и не могут быть сжаты. Термин «сжатый приватный ключ» на самом деле означает "приватный ключ, из которого могут быть получены сжатые приватные ключи," в то время как "несжатый приватный ключ" на самом деле означает "приватный ключ, от которого могут быть получены несжатые публичные ключи." Чтобы избежать дальнейшей путаницы "WIF-сжатый" или "WIF"

мы будем называть только формат экспорта, не относящийся к приватному ключу.

Помните, что эти форматы *не* взаимозаменяемы. В более новых кошельках, которые используют сжатые публичные ключи, приватные ключи будут экспортированы только сжатыми (с префиксом K или L). Если кошелек более старой реализации и не использует сжатые публичные ключи, приватные ключи будут экспортированы в WIF (с префиксом 5). Здесь задача просигнализировать импортирующему кошельку, что он должен просканировать блокчейн на предмет сжатых или несжатых публичных ключей и адресов.

Если Биткоин-кошелек поддерживает сжатые публичные ключи, он будет их использовать во всех транзакциях. Сжатые публичные ключи будут использоваться для получения Биткоин-адреса, которые, в свою очередь, будут использованы в транзакциях. При экспорте приватных ключей из более нового кошелька, поддерживающего сжатые публичные ключи, используется модифицированный Wallet Import Format с добавлением одного байта суффикса 01 к закрытому ключу. Результирующий приватный ключ в кодировке Base58Check называется "Сжатый WIF" и начинается с буквы K или L, а не с "5", как в случае с WIF-кодированными (несжатыми) ключами из более старых кошельков.

В [Пример: один и тот же ключ в разных форматах](#) показан тот же ключ, закодированный в WIF и WIF-сжатом форматах.

Table 4. Пример: один и тот же ключ в разных форматах

Формат	Приватный ключ
Шестнадцатичный	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbkneyhfsYB1Jcn
Шестнадцатичный и сжатый	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD_01_
Сжатый WIF	KxFC1jmwWCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

ТИП

"Сжатые приватные ключи" не существуют! Они не сжаты; скорее, WIF-сжатый формат сигнализирует, что они должны быть использованы только для получения сжатых публичных ключей и соответствующих им адресов Биткоин. Парадоксально, но "WIF-сжатый" приватный ключ на один байт длиннее, потому что имеет дополнительный суффикс 01, отличающий его от "несжатого".

Реализация ключей и адресов на Python

Самая полная библиотека Bitcoin для Python написана Виталиком Бутериным и может быть найдена по адресу [pybitcointools](#). В [Генерация ключа и адреса и форматирование при помощи библиотеки pybitcointools](#) мы используем библиотеку [pybitcointools](#) (импортирована как

"bitcoin") для генерации и показа ключей и адресов в различных форматах.

Example 4. Генерация ключа и адреса и форматирование при помощи библиотеки pybitcointools

```
import bitcoin

# Generate a random private key
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

# Convert private key to WIF format
wif_encoded_private_key = bitcoin.encode_privkey(decoded_private_key, 'wif')
print "Private Key (WIF) is: ", wif_encoded_private_key

# Add suffix "01" to indicate a compressed private key
compressed_private_key = private_key + '01'
print "Private Key Compressed (hex) is: ", compressed_private_key

# Generate a WIF format from the compressed private key (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

# Multiply the EC generator point G with the private key to get a public key point
public_key = bitcoin.fast_multiply(bitcoin.G, decoded_private_key)
print "Public Key (x,y) coordinates is:", public_key

# Encode as hex, prefix 04
hex_encoded_public_key = bitcoin.encode_pubkey(public_key, 'hex')
print "Public Key (hex) is:", hex_encoded_public_key

# Compress public key, adjust prefix depending on whether y is even or odd
(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

# Generate bitcoin address from public key
```

```
print "Bitcoin Address (b58check) is:", bitcoin.pubkey_to_address(public_key)

# Generate compressed bitcoin address from compressed public key
print "Compressed Bitcoin Address (b58check) is:", \
    bitcoin.pubkey_to_address(hex_compressed_public_key)
```

В [Запускаем key-to-address-ecc-example.py](#) показан вывод после запуска этого кода.

Example 5. Запускаем key-to-address-ecc-example.py

[Сценарий демонстрирует математику эллиптических кривых, используемую для ключей в Биткоин](#) — это еще один пример применения библиотеки Python ECDSA для математики эллиптических кривых без использования каких-либо специализированных библиотек.

Example 6. Сценарий демонстрирует математику эллиптических кривых, используемую для ключей в Биткоин

```
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 = 0x000000000000000000000000000000000000000000000000000000000000007L
_a = 0x00000000000000000000000000000000000000000000000000000000000000L
_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)

    # Collect 256 bits of random data from the OS's cryptographically secure random
    generator
    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')

# Generate a new private key.
secret = random_secret()
print "Secret: ", secret

# Get the public key point.
point = secret * generator
print "EC point:", point

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

# Given the point (x, y) we can create the object using:
point1 = ecdsa.ellipticcurve.Point(curve, point.x(), point.y(), ec_order)
assert point1 == point

```

В [Установка библиотеки Python ECDSA и запуск сценария es_math.py](#) показан вывод после запуска этого сценария.

NOTE

В приведенном выше примере используется функция `os.urandom`, которая вызывает криптографически безопасный генератор случайных чисел (CSRNG), предоставляемый операционной системой. В случае UNIX-подобной операционной системы, такой как Linux, он читает из `/dev/urandom`; а в случае Windows, происходит вызов `CryptGenRandom()`. Если подходящий источник случайности не найден, будет брошено исключение `NotImplementedError`. В то время как генератор случайных чисел, используемый здесь, в демонстрационных целях, это *не* подходит для генерации ключей Bitcoin, так как его безопасность недостаточна.

```
$ # Устанавливаем PIP -- менеджер пакетов Python
$ sudo apt-get install python-pip
$ # Устанавливаем библиотеку ECDSA
$ sudo pip install ecdsa
# $ Запускаем сценарий
$ python es-math.py
Secret:
38090835015954358862481132628887443905906204995912378278060168703580660294000
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

Кошельки

Кошельки—это контейнеры для частных ключей, как правило, реализуются в виде структурированных файлов или простых баз данных. Еще один метод создания ключей называется *детерменистическая генерация ключей*. Здесь, каждый новый приватный ключ получается использованием односторонней хэш-функции от предыдущего закрытого ключа, связывая их в последовательности. До тех пор пока вы можете воссоздать эту последовательность, вам нужен только первый ключ (известный как *зерно* или *мастер-ключ*). В этом разделе мы рассмотрим различные методы генерации ключей и структур кошелька, построенных вокруг этого.

ТИП

Биткоин-кошельки пользователей хранят пары ключей, а не монеты (см [Приватные и публичные ключи](#)). Пользователи подписывают транзакции ключами, таким образом доказывая права владения выходами транзакций (их монетами). Монеты хранятся в блокчейне в виде транзакций-выходов (часто обозначаемых как `vout` или `txout`).

Недетерминированные (случайные) кошельки

В первых Биткоин-клиентах, кошельками являлись просто коллекции случайно сгенерированных частных ключей. Этот тип кошелька называется *недетерминированным кошельком Тип-0*. Например, клиент Bitcoin Core регенерирует 100 случайных частных ключей при первом запуске, и генерирует новые по мере необходимости. Этот тип кошелька прозвали "Просто Куча Ключей", или ПКК ("Just a Bunch Of Keys," JBOK) неудобны для резервного копирования, импорта/экспорта, и им на замену приходят детерминированные кошельки. Недостатком случайных ключей является то, что если вы создадите их много, то придется хранить копии их всех, а значит часто делать резервные копии кошелька, иначе, в случае безвозвратной потери ключа, средства будут навсегда потеряны. Подобное неудобство

влечет за собой переиспользование адресов, что уменьшает конфиденциальность, позволяя строить ассоциации между различными адресами и транзакциями. Хотя клиент Bitcoin Core включает в себя недетерминированный кошелек Типа-0, его использование не рекомендуется разработчиками Bitcoin Core. На рис. [Недетерминированный кошелек Типа-0: коллекция случайно сгенерированных ключей](#) показан недетерминированный кошелек с коллекцией случайных ключей.

Детерминированные (с зерном) кошельки

Детерминистические, или кошельки "с зерном"—это такие, в которых приватные ключи происходят от общего зерна через использование односторонней хэш-функции. Само по себе зерно — это случайное число, в сочетании с другими данными, такими как номер индекса или "кодом цепи" (см. [Иерархические детерминированные кошельки \(BIP0032/BIP0044\)](#)) позволяет получить приватные ключи. В детерминированном кошельке, для восстановления всех ключей достаточно знать зерно, а следовательно, достаточно одного единственного резервного копирования во время его создания. Зерна также достаточно для экспорта или импорта кошелька, что позволяет легко мигрировать все ключи пользователя между различными реализациями кошелька.

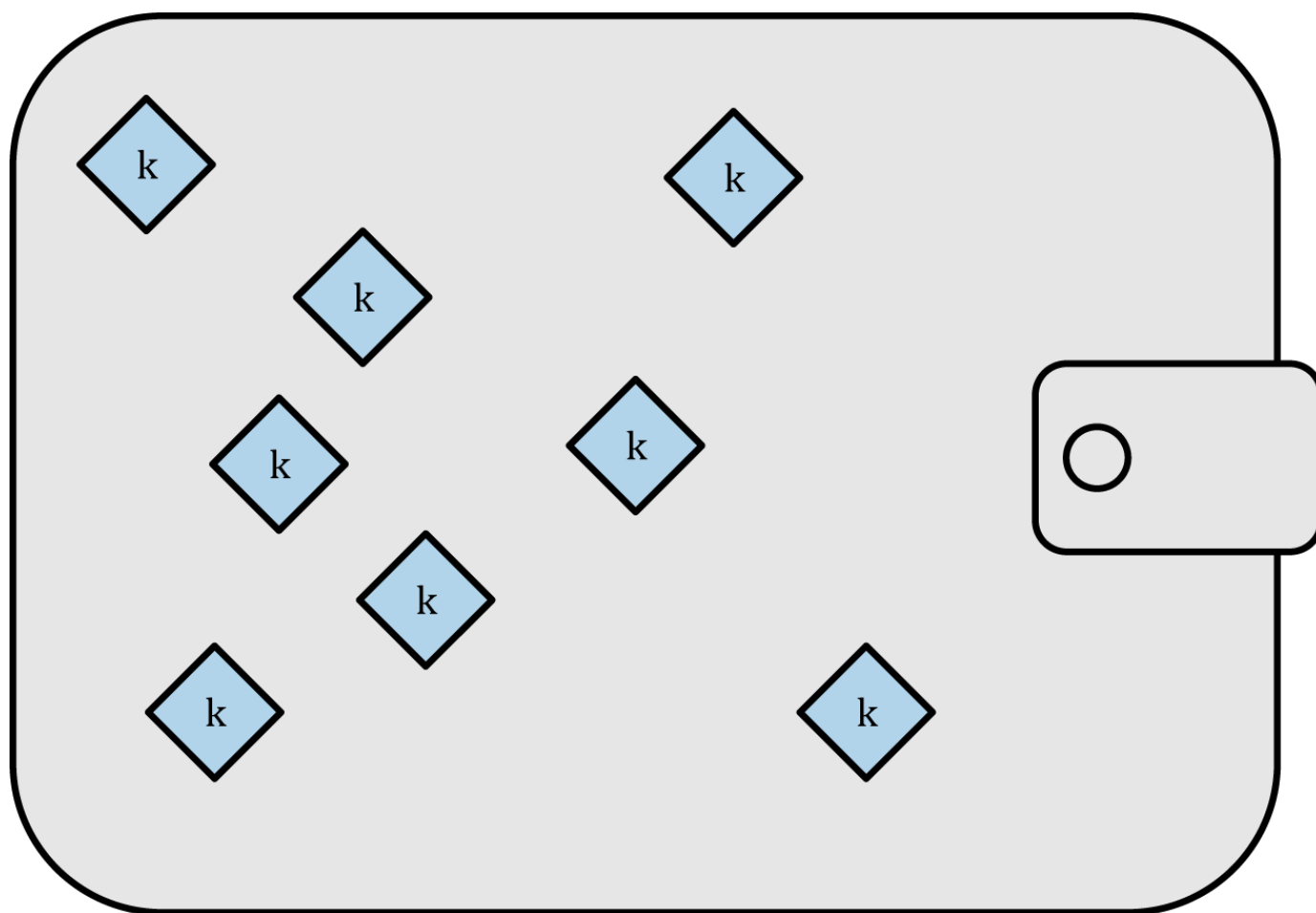


Figure 8. Недетерминированный кошелек Типа-0: коллекция случайно сгенерированных ключей

Мнемонические кодовые слова

Мнемонические коды — это последовательности английских слов, которые представляют собой (кодируют) случайное число, которое используется в качестве зерна для детерминированного кошелька. Последовательности слов достаточно, чтобы заново создать зерно, и из него уже воссоздать бумажник и все производные ключи. Приложение кошелька, реализующее детерминированные кошельки с мнемоническим кодом покажет пользователю последовательность из от 12 до 24 слов при создании кошелька. Это последовательность слов является резервной копией кошелька и может быть использована для восстановления и повторного создания всех ключи в том же или любом другом совместимом приложении кошелька. Мнемонические кодовые слова читаются и транскрибируются легче по сравнению со случайной последовательностью чисел.

Мнемоник коды описаны в Предложении Улучшения Биткоин номер 39 (см. [\[bip0039\]](#)), которое в настоящее время находится в статусе черновика. Обратите внимание, что BIP0039 является проектом предложения, а не стандартом. В частности, есть другой стандарт, с другим набором слов, используемых в кошельке Electrum, предшествовавшим BIP0039, BIP0039 используется кошельке Trezor и некоторых других, несовместимых с реализацией в Electrum.

В BIP0039 описано создание мнемонического кода и зерна в следующем виде:

1. Создать случайную последовательность (энтропию) длиной от 128 до 256 бит.
2. Создать контрольную сумму случайной последовательности, взяв первые несколько битов из ее SHA256 хэша.
3. Добавить контрольную сумму в конец случайной последовательности.
4. Разделить последовательность на части длиной в 11 бит, и использовать их в качестве индекса по словарю из 2048 предопределенных слов.
5. Полусить от 12 до 24 слов, представляющих собой мнемонический код.

В таблице [Мнемонические коды: энтропия и длина слов](#) показано соотношение между размером данных энтропии и длиной мнемонических кодов в словах.

Table 5. Мнемонические коды: энтропия и длина слов

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

Мнемонический код представляет от 128 до 256 бит, которые используются для получения более длинного (512-битного) зерна за счет использования функции удлинения ключа PBKDF2. Полученное зерно используется для создания детерминистического кошелька и производных от него ключей.

В таблицах `<xref linkend="table_4-6" xrefstyle="select: labelnumber"/>` и `<xref linkend="table_4-7" xrefstyle="select: labelnumber"/>` показаны некоторые примеры мнемонических кодов и зерен, которые они производят.

Table 6. Мнемонический код из .128 битной энтропии и результирующее зерно

Энтропия на входе (128 бит)	0c1e24e5917779d297e14d45f14e1a1a
Мнемонический код (12 слов)	army van defense carry jealous true garbage claim echo media make crunch
Зерно (512 битов)	3338a6d2ee71c7f28eb5b882159634cd46a898463e9 d2d0980f8e80dfbba5b0fa0291e5fb88 8a599b44b93187be6ee3ab5fd3ead7dd646341b2cd b8d08d13bf7

Table 7. 256-битный энтропийный мнемонический код и результирующее зерно

Вход энтропии (256 битов)	2041546864449caff939d32d574753fe684d3c947c33 46713dd8423e74abcf8c
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
Зерно (512 битов)	3972e432e99040f75ebe13a660110c3e29d131a2c80 8c7ee5f1631d0a9777fcf473bee22 fce540af281bf7cdeade0dd2c1c795bd02f1e4049e20 5a0158906c343

Иерархические детерминированные кошельки (VIP0032/VIP0044)

Детерминистические кошельки были разработаны, чтобы можно было легко получить много ключей из одного «зерна». Наиболее продвинутой формой детерминистических кошельков является *иерархический детерминистический кошелек* или *HD-кошелек*, описанный в стандарте VIP0032. Иерархические детерминистические кошельки содержат ключи в виде древовидной структуры, так что из родительского ключа можно вывести последовательность производных ключей, от каждого из которых, в свою очередь, также получается последовательность производных ключей и так далее без ограничения глубины вложенности. Эта древовидная структура показана на [Иерархическую детерминистический кошелек Тип-2: дерево ключей, генерируемых из единственного зерна](#).

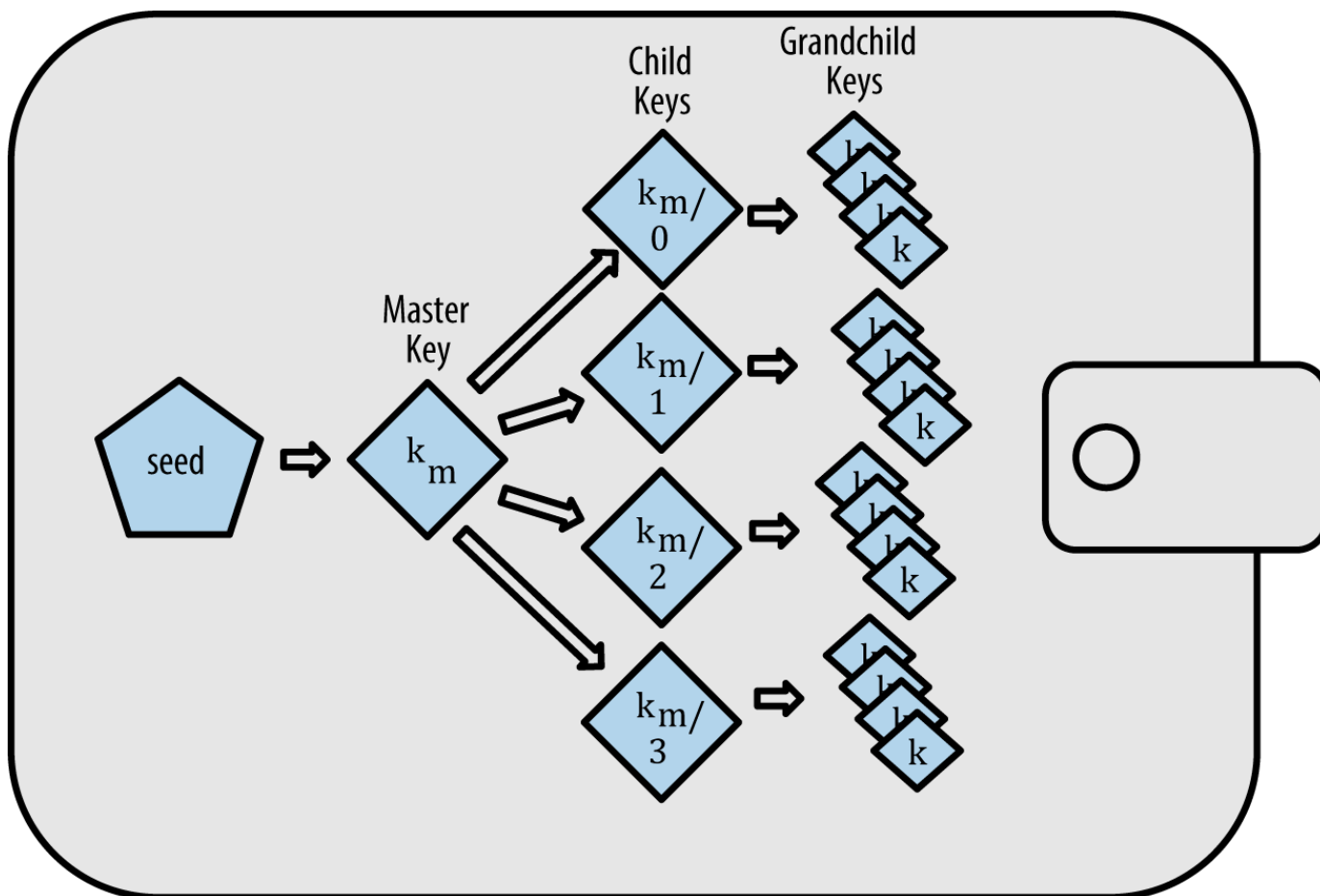


Figure 9. Иерархическую детерминистический кошелек Тип-2: дерево ключей, генерируемых из единственного зерна

ТИП

Если вы разрабатываете Биткоин-кошелек, его следует реализовать в виде HD-кошелька согласно стандартам BIP0032 и BIP0044.

HD-кошельки предлагают два основных преимущества по сравнению с недетерминированными. Во-первых, структура дерева может быть использована для дополнительной организации, например, когда конкретная ветвь ключей используется для получения входящих платежей, а другая ветвь используется для получения сдачи для исходящих платежей. Ветвления ключей также могут быть использованы в корпоративной среде для различных отделов, филиалов, конкретных функций, или категорий бухгалтерского учета.

Второе преимущество HD-кошельков в том, что их пользователи могут создавать последовательности открытых ключей, не имея доступа к соответствующим приватным ключам. Это позволяет использовать HD-кошельки на небезопасных серверах или в режиме только приема средств с выдачей нового публичного ключа для каждой новой транзакции. Публичные ключи не должны быть предварительно загружены или получены заранее и сервер не содержит приватные ключи, которые могут использоваться для траты средств.

Создание HD-кошелька при помощи зерна

HD-кошельки создаются из единственного *корневого зерна*, которое представляет собой 128-, 256-, или 512-битное случайное число. Все остальное в HD-кошельке детерминистически производно от этого корневого зерна, которое делает возможным воссоздать весь HD-кошелек из этого зерна в любом совместимом ПО. Это позволяет просто осуществлять создать резервное копирование, восстановление, экспорт, и импорт кошельков, содержащих тысячи или даже миллионы ключей. Корневое зерно наиболее часто задается в виде *мнемонической последовательности слов*, как описано в предыдущем разделе [Мнемонические кодовые слова](#).

Процесс создания мастер-ключей и мастер-цепочки для HD-кошелька показан на [Срздание мастер-ключей и кода цепочки из корневого зерна](#).

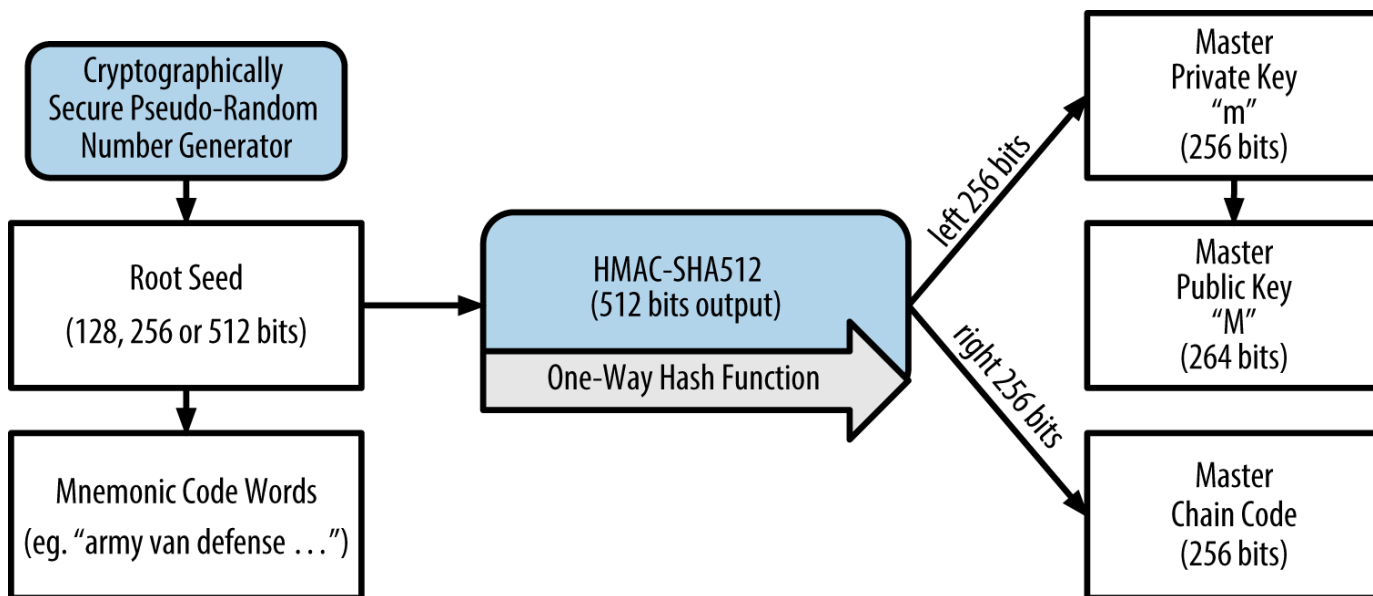


Figure 10. Срздание мастер-ключей и кода цепочки из корневого зерна

Корневое зерно пропускается через алгоритм HMAC-SHA512, и полученный хэш используется для создания *главного приватного ключа* (m) и *главной цепочки кодов*. Главный приватный ключ (m) используется для генерации соответствующего главного публичного ключа (M), используя обычный процесс умножения на эллиптических кривых $m * G$, описанный ранее в этой главе. Код цепи используется для введения энтропии в функцию, которая создает производные ключи из родительских ключей, как мы увидим в следующем разделе.

Производные дочерние приватные ключи

Для получения дочерних ключей из родительских, иерархические детерминистические кошельки используют функцию *деривации дочерних ключей* (ДДК).

Функции деривации дочерних ключей основаны на односторонней хеш-функции, комбинирующей:

- Родительский приватный или публичный ключ (несжатый ECDSA ключ)
- Зерно, называемое кодом цепочки (256 бит)

- Порядковым номером (32 бита)

Код цепи используется для введения случайных данных в процесс, так чтобы индекс не был достаточным для получения других соседних дочерних ключей. Таким образом, знание дочернего ключа не позволяет найти его братьев и сестер, при условии, что код цепи не известен. Исходное зерно кода цепи (в корне дерева) получен случайно, в то время как последующие коды цепи получены из родительского кода цепи.

Эти три элемента объединены и хэшированы для генерации дочерних ключей следующим образом.

Родительский публичный ключ, код цепи и порядковый номер объединяются и хэшируются алгоритмом HMAC-SHA512, получая на выходе 512-битный хэш. Полученную хэш разделяется на две половины. Правые 256 бит хэш-кода становятся дочерним кодом цепи. Левые 256 бит хэша и индексный номер добавляются к родительскому приватному ключу для получения дочернего приватного ключа. В [Расширение родительского приватного ключа для получения дочернего приватного ключа](#) это проиллюстрировано для получения 0-ого (первого по порядку) производного ключа.

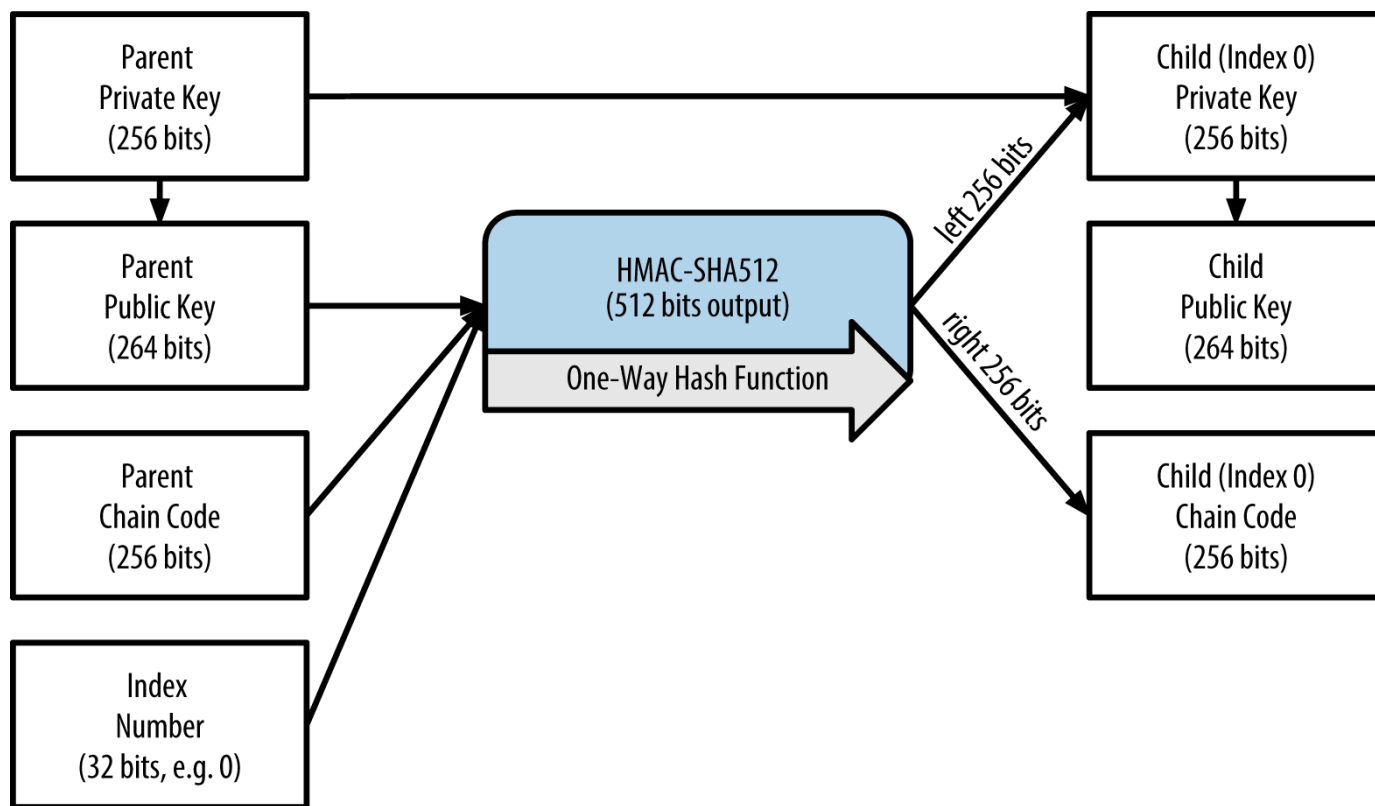


Figure 11. Расширение родительского приватного ключа для получения дочернего приватного ключа

Изменение порядкового номера позволяет получить другие дочерние ключи в последовательности, например Дочерний Ключ 0, Дочерний Ключ 1, Дочерний Ключ 2, и т.д. Каждый родительский ключ может иметь 2 млрд производных ключей.

Повторяя процесс одним уровнем ниже по дереву, каждый дочерний ключ может, в свою

очередь стать родительским для своих собственных дочерних ключей, в бесконечном ряду поколений.

Использование производных ключей

Дочерние приватные ключи неотличимы от недетерминированных (случайных) ключей. Поскольку функция порождения является односторонней функцией, дочерний ключ не может быть использован для нахождения родительского ключа. Дочерний ключ также не может быть использован для нахождения соседних с ним ключей. Если у вас есть $n_{\text{ый}}$ потомок, вы не можете найти его братьев и сестер, таких, как потомок $n-1$ потомок $n+1$, или каких-либо других потомков из последовательности. Только родительский ключ и код цепи могут произвести все дочерние ключи. Без дочернего кода цепи дочерний ключ так же не может быть использован для получения внучатых ключей. Для создания новой ветки последовательности внучатых ключей потребуется как дочерний приватный ключ, так и дочерний код цепи.

Так для чего можно использовать дочерний приватный ключ сам по себе? Он может быть использован для получения публичного ключа и Биткоин-адреса, а также для подписывания транзакций от имени этого адреса.

ТИП

Дочерний приватный ключ, соответствующий публичный ключ, а также Биткоин-адрес все неотличимы от ключей и адреса, созданных случайно. Тот факт, что они являются частью последовательности не заметен за пределами HD-кошелька, который их создал. После создания они "работают" как "обычные" ключи.

Расширенные ключи

Как мы видели ранее, функция деривации ключей может быть использована для создания дочерних ключей на любом уровне дерева с помощью трех компонент: ключа, кода цепи и порядкового номера. Комбинация ключа и кода цепи называется *расширенным ключем*. Термин "расширенный ключ" может также рассматриваться как "расширяемый ключ" в том смысле, что с его помощью можно получать производные дочерние ключи.

Расширенные ключи хранятся и выглядят просто как конкатенация 256-битного ключа и 256-битного кода цепи в 512-битовую последовательность. Есть два типа расширенных ключей. Расширенный приватный ключ является комбинацией приватного ключа и кода цепи и может быть использован для получения дочерних приватных ключей (и соответствующие им публичные ключи). Расширенный публичный ключ — это публичный ключ и код цепи, которые могут быть использованы для получения дочерних публичных ключей, как описано в [Создание публичного ключа](#).

Расширенный ключ можно представить в виде основания какой-либо ветки в древовидной структуре HD-кошелька. Обладая основанием можно получить остальную часть ветки. Расширенный приватный ключ может создать полную ветвь, в то время как расширенный публичный ключ может создать только ветку публичных ключей.

ТИП

Расширенный ключ состоит из приватного или публичного ключа и кода цепи. Расширенный ключ может создавать потомков в собственной ветке на общем дереве. Поделившись расширенным ключом можно дать доступ к ветке целиком.

Расширенные ключи кодируются с помощью Base58Check, что позволяет легко обмениваться ими между различными BIP0032-совместимыми кошельками. Для кодирования расширенных ключей в Base58Check используется специальный номер версии, который после кодирования выглядит как префикс "xprv" или "xpub". Так как расширенный ключ имеет длину 512 или 513 бит, результат кодирования намного длиннее, чем те Base58Check-закодированные строки, которые мы уже видели ранее.

Вот пример расширенного приватного ключа, закодированного в Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMK  
Uga5biW6Hx4tws2six3b9c
```

Вот соответствующий расширенный публичный ключ, также закодированный в Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWHPGq6Qv5cmNfi7cS5mtjJ2tgyeQbBs2UAR6KECeeMVKZBPLrtJunSDMst  
weyLXhRgPxdp14sk9tJPW9
```

Производные публичные ключи

Как упоминалось ранее, очень полезное свойство иерархических детерминированных кошельков состоит в том, что с их помощью можно получать дочерние публичные ключи без обладания приватными ключами. Это дает нам два способа получения дочернего открытого ключа: либо из дочернего приватного ключа, либо непосредственно из родительского публичного ключа.

Расширенный публичный ключ может быть использован, поэтому, в качестве производного для всех публичных ключей (и только публичных) в этой ветке структуры HD-кошелька.

Эта удобная возможность позволяет создавать очень безопасные среды, где сервер или приложение будет иметь доступ к публичному расширенному ключу и не иметь приватных ключей вообще. Таким образом можно производить бесконечное число публичных ключей и Биткоин-адресов, но нельзя потратить деньги, присланные на эти адреса. Между тем, на другом, более безопасном сервере, расширенный приватный ключ может использоваться для получения всех соответствующих приватных ключей, чтобы можно было подписывать транзакции и тратить деньги.

Одно из распространенных применений этого решения является установка расширенного публичного ключа на веб-сервере, обслуживающем приложение электронной коммерции. Веб-сервер может использовать функцию деривации публичного ключа для создания нового Биткоин-адреса для каждой транзакции. Веб-сервер не будет содержать никаких приватных

ключей, которые будут уязвимы для кражи. Без HD-кошелька, единственный способ сделать то же самое это сгенерировать тысячи Биткоин-адресов на отдельном защищенном сервере и затем загрузить их на сервер электронной коммерции. Такой подход является громоздким и требует постоянного обслуживания, так как надо гарантировать, что на сервере электронной коммерции не "кончатся" ключи.

Еще одно распространенное применение этого решения—это холодные и аппаратные кошельки. При таком сценарии, расширенный приватный ключ может храниться на листе бумаги или в электронном устройстве (например таком, как Trezor), в то время как расширенный публичный ключ может быть онлайн. Пользователь может создавать адреса "для получения" по желанию, в то время как приватные ключи надежно хранятся в офлайне. Для траты средств, пользователь может использовать расширенный приватный ключ при помощи офлайн-ового Биткоин-клиента или подписать транзакции при помощи аппаратного устройства-кошелька (например, Trezor). На [Расширение родительского публичного ключа для создания дочернего публичного ключа](#) проиллюстрирован механизм расширения родительского публичного ключа для получения дочерних публичных ключей.

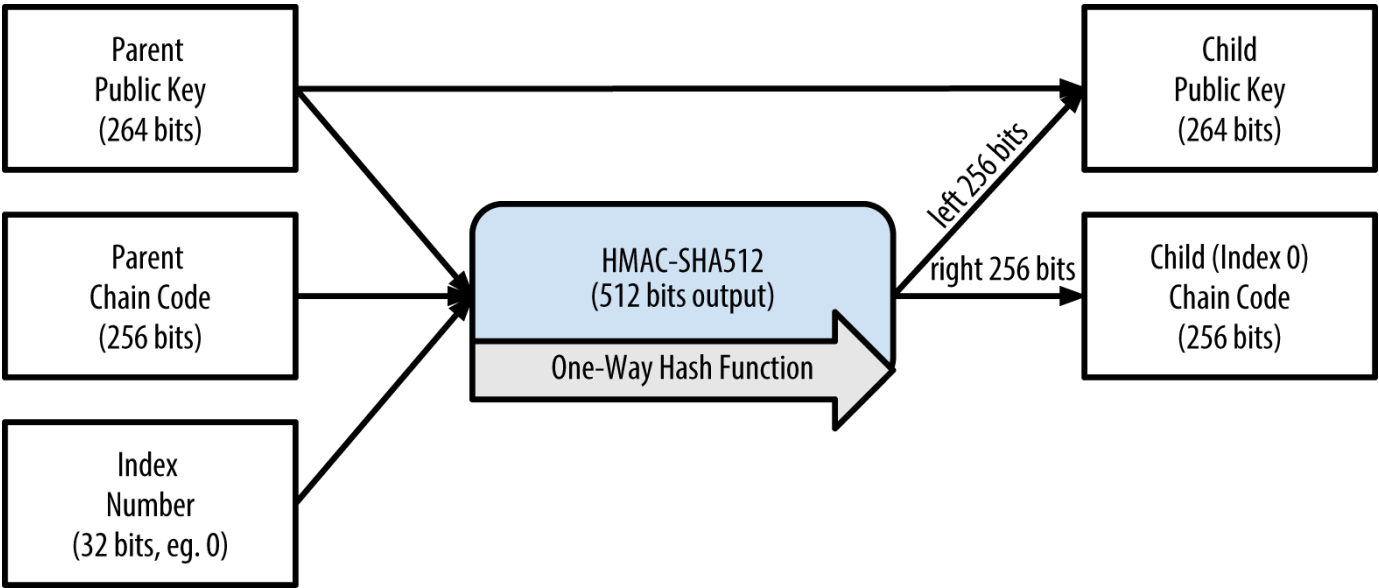


Figure 12. Расширение родительского публичного ключа для создания дочернего публичного ключа

Укрепленные производные дочерние ключи

Возможность получения ветки производных публичных ключей из расширенного публичного ключа очень полезна, однако в ней заложен потенциальный риск. Доступ к расширенному публичному ключу не дает доступа к дочерним приватным ключам. Тем не менее, так как расширенный публичный ключ содержит код цепи, если дочерний публичный ключ известен, или как-то утек, он может использоваться с кодом цепи для получения всех остальных дочерних приватных ключей. Единственный утекший приватный ключ, совместно с родительским кодом цепи, раскрывает все приватные ключи всех детей. Хуже того, дочерний приватный ключ вместе с родительским кодом цепи могут быть использованы для вывода родительского приватного ключа.

Чтобы противостоять этой опасности, в HD-кошельках используется альтернативная функция деривации, называемая *укрепленная деривация*, которая "ломает" отношение между родительским публичным ключом и дочерним кодом цепи. Функция укрепленной деривации использует родительский приватный ключ для получения дочернего кода цепи, вместо родительского публичного ключа. Это создает "брандмауэр" в последовательности родитель/потомок, с кодом цепи, который не может быть использован для компрометации родительского или родственного приватного ключа. Укрепленная функция деривации выглядит почти идентично обычной функции выведения дочернего приватного ключа, за исключением того, что родительский приватный ключ используется в качестве входа для хэш-функции, вместо родительского публичного ключа, как показано на схеме [Укрепленный вывод дочернего ключа; не включает родительский публичный ключ](#).

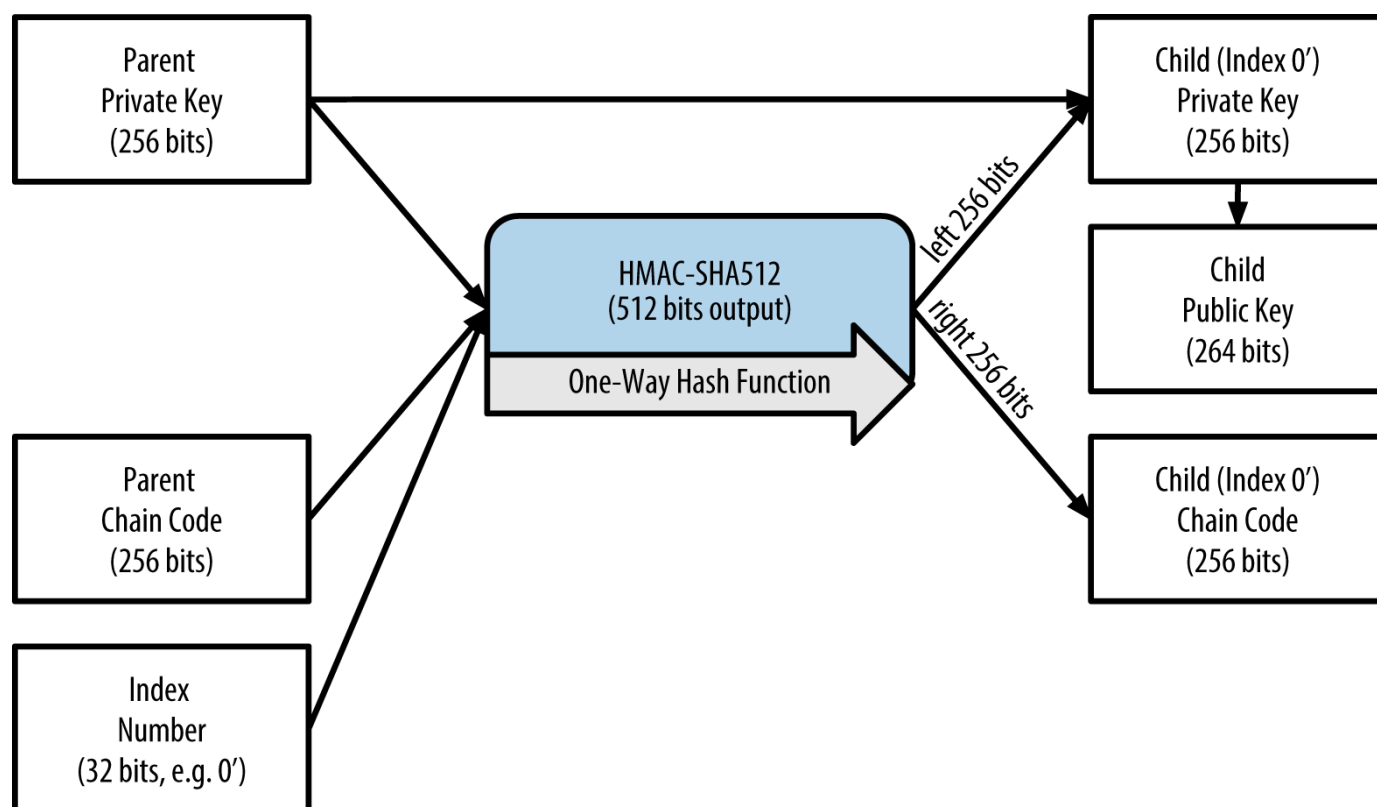


Figure 13. Укрепленный вывод дочернего ключа; не включает родительский публичный ключ

Когда используется укрепленная функция деривации приватных ключей, результирующий дочерний приватный ключ и код цепи полностью отличается от того, который получается в результате выполнения обычной функции деривации. Результирующая "ветка" ключей может быть использована для получения расширенных публичных ключей, которые не были бы уязвимы, так как они содержат код цепи, который не может использоваться для выявления каких-либо приватных ключей. Укрепленная деривация, следовательно, используется для создания "разрыва" в дереве выше уровня, где используются расширенные публичные ключи.

Простыми словами, если вы хотите использовать удобство расширенного публичного ключа по деривации ветви публичных ключей, не подвергая себя риску утечки кода цепи, вы должны произвести его от укрепленного прородителя, а не от обычного. Лучше всего, если потомки -1-го уровня всегда получены в результате укрепленной деривации, чтобы предотвратить

компрометацию мастер-ключа.

Порядковый номер для обычной и укрепленной деривации

Порядковый номер, используемый в функции деривации — это 32-разрядное целое число. Чтобы легко отличать ключи, полученные с помощью обычной функции деривации ключей от ключей, полученных при помощи укрепленной функции, этот порядковый номер разделен на два диапазона. Порядковые номера между 0 и $2^{31}-1$ (от 0x0 до 0x7FFFFFFF) используются *только* для нормальной деривации. Номера между 2^{31} и $2^{32}-1$ (от 0x80000000 до 0xFFFFFFFF) используются *только* для укрепленной деривации. Таким образом, если порядковый номер меньше 2^{31} , это означает, что дочерний ключ обычный, в то время как, если номер равен или больше 2^{31} , дочерний ключ укрепленный.

Для того, чтобы порядковый номер легче читаться, для укрепленных дочерних ключей он начинается с нуля, но с символом апострофа. Поэтому первый обычный дочерний ключ отображается как 0, в то время как первый укрепленный дочерний ключ (индекс 0x80000000) отображается как <markup>0'</markup>. Далее второй укрепленный ключ имел бы номер 0x80000001 и отображался бы как 1', и так далее. Когда вы видите порядковый номер HD-кошелька i, это означает $2^{31}+i$.

Идентификационный ключ (путь) в HD-кошельке

Ключи внутри HD-кошелька идентифицируются с помощью "пути", где каждый уровень дерева отделяется символом косой черты (см [Примеры путей HD-кошельков](#)). Приватные ключи, полученные от главного приватного ключа начинаются с "m". Публичные ключи, полученные от главного публичного ключа начинаются с "M". Таким образом, первый дочерний приватный ключ, полученный от главного имеет путь m/0. Первый дочерний публичный ключ имеет путь M/0. Второй внучатый ключ, полученный от первого дочернего имеет путь m/0/1, и так далее.

"Родословную" ключа можно прочесть справа налево, пока не будет достигнут главный ключ, из которого он был получен. Например, идентификатор m/x/y/z описывает ключ, который является z-ым дочерним для m/x/y, который, в свою очередь, является y-ым дочерним для m/x, а тот x-ый по счету для m.

Table 8. Примеры путей HD-кошельков

Путь	Описание ключа
m/0	Первый (0) дочерний приватный ключ из приватного мастер-ключа (m)
m/0/0	Приватный ключ первого внучатого ключа от первого дочернего (m/0)
m/0'/0	Первый обычный внучатый ключ от первого укрепленного дочернего ключа (m/0')

Путь	Описание ключа
m/1/0	Первый внучатый приватный ключ от второго дочернего ключа (m/1)
M/23/17/0/0	Первый праправнучатый публичный ключ от первого правнучатого от 18-го внучатого от 24-го дочернего ключа

Навигация по древовидной структуре HD-кошелька

Структура дерева HD-кошелька предлагает большую гибкость. Каждый родительский расширенный ключ может иметь 4 млрд непосредственных потомков: 2 миллиарда обычных и 2 миллиарда укрепленных. Каждый из этих потомков может иметь еще 4 млрд потомков, и так далее. Дерево может иметь любую желаемую глубину. При всей этой гибкости, однако, ориентироваться в бесконечном дереве довольно трудно. Особенно трудно переносить HD кошелек между реализациями, так как возможности внутреннего разделения на ветки и подветки поистинне бесконечны.

Два предложения из Bitcoin Improvement Proposals (BIP) предлагают решения для этой сложности в виде предложения стандартов структуры дерева HD-кошелька. BIP0043 предлагает использование номера первого укрепленного ключа в качестве специального идентификатора, который выражает "назначение" структуры дерева. HD-кошелек на основе BIP0043 должен использовать только одну ветвь первого уровня дерева, с порядковым номером, идентифицирующим структуру и пространство имен остальной части дерева, определяя ее назначение. Например, HD-кошелек, использующий только ветку m/i'/ предназначен для определенной конкретной цели, и эта цель определяется порядковым номером "i".

Расширяя эту спецификацию, BIP0044 предлагает мультиаккаунтную структуру в качестве "предназначения" номер 44' внутри BIP0043. Все HD-кошельки, следующие структуре BIP0044 используют только одну ветвь дерева: m/44'/.

BIP0044 определяет структуру, состоящую из пяти предопределенных уровней дерева:

m / назначение' / тип_монеты' / счет' / сдача / порядковый_номер_адреса

"Назначение" первого уровня всегда установлено в 44'. Второй уровень "coin_type" специфицирует криптовалюту, что позволяет иметь мультивалютные HD-кошельки, где каждая валюта получает свое собственное поддерево под вторым уровнем. В данный момент определены три валюты: Bitcoin m/44'/0', Bitcoin Testnet `<markup>m/44'/1'</markup>`; и Litecoin `<markup>m/44'/2'</markup>`.

Третий уровень дерева — это "счет", который позволяет пользователям дополнительно разделить свои кошельки на отдельные логические субсчета, для учета или организационных целей. Например, HD-кошелек может содержать два "счета": `<markup>m/44'/0'/0'</markup>` and `<markup>m/44'/0'/1'</markup>`. Каждый счет служит корнем

своего собственного поддерева.

На четвертом уровне, "сдачи", дерево раздваивается на два поддерева, одно для создания приемных адресов и одно для создания адресов для сдачи. Следует отметить, что в то время как предыдущие уровни использовались для укрепленных производных, на этом уровне используются нормальные производные. Это должно позволить экспортировать расширенные открытые ключи с этого уровня для использования в небезопасной среде. Используемые адреса производны от четвертого уровня, что делает пятый уровень дерева "address_index." Например, третий приемный адрес для Биткойн платежей в первичном счете будет М/44'/0'/0'/0/2.. В таблице [Примеры структуры HD-кошелька, описанной в ВІР0044](#) показано несколько примеров.

Table 9. Примеры структуры HD-кошелька, описанной в ВІР0044

Путь	Описание ключа
М/44'/0'/0'/0/2	Третий приемный публичный ключ для главного счета Биткойн
М/44'/0'/3'/1/14	Публичный ключ пятнадцатого адрес для сдачи четвертого счета Биткойн
м/44'/2'/0'/0/1	Второй приватный ключ от главного счета в сети Litecoin, для подписания сделок

Эксперименты с HD-кошельками с помощью Bitcoin Explorer

С помощью инструмента командной строки Bitcoin Explorer, с которым читатель познакомился в [\[ch03_bitcoin_client\]](#), можно экспериментировать с созданием и расширением детерминированных ключей ВІР0032, а также просто выводить их в различных форматах:

```
$ bx seed | bx hd-new > m # создаем новый приватный мастер-ключ из зерна и
сохраним в файле с названием "m"
$ cat m # показать расширенный приватный мастер-ключ
xprv9s21ZrQH143K38iQ9Y5p6qoB8C75TE71NfpyQPdfGvzghDt39DHPFpovvtWZaRgY5uPwV7RpEgHs7cvdg
fiSjLjjbuGKGcjRyU7RGGSS8Xa
$ cat m | bx hd-public # генерируем расширенный публичный ключ M/0
xpub67xprozc8pe95XVuZLHXZeG6XWHPGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunS
DMstweyLXhRgPxdp14sk9tJPW9
$ cat m | bx hd-private # генерируем расширенный приватный ключ m/0
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6G
oNMKUga5biW6Hx4tws2six3b9c
$ cat m | bx hd-private | bx hd-to-wif # показать приватный ключ m/0 в формате
WIF
L1pbvV86crAGoDzmqmY85xURkz3c435Z9nirMt52UbnGjYMzKBUN
$ cat m | bx hd-public | bx hd-to-address # показать Биткоин-адрес M/0
1CHCnCjgMnb6digimckNQ6TBVcTWBAмPHK
$ cat m | bx hd-private | bx hd-private --index 12 --hard | bx hd-private --index 4 #
генерируем m/0/12'/4
xprv9yL8ndfdPVeDWJenF18oiHguRUj8jHmVrqqD97YQHeTcR3LCeh53q5PXPkLsy2kRaqqwoS6YZBLatRZRy
UeAkRPe1kLR1P6Mn7jUrXFquUt
```

Продвинутые ключи и адреса

В следующих разделах мы рассмотрим продвинутые формы ключей и адресов, например, зашифрованные приватные ключи, адреса сценариев и многоподписные адреса, адреса тщеславия и бумажные кошельки.

Шифрованные приватные ключи (BIP0038)

Приватные ключи должны оставаться в тайне. Необходимость *сокрытия* закрытых ключей общеизвестна, но достаточно трудно реализуема на практике, потому что он конфликтует с не менее важной целью безопасности — *доступностью*. Сохранять приватный ключ в секрете гораздо труднее, если вам приходится делать его резервные копии. Хранение приватного ключа в зашифрованном паролем кошельке может быть безопасным, но кошелек должен быть резервно копироваться. Время от времени, пользователям приходится переносить ключи из одного кошелька в другой например для замены или обновления программного обеспечения. Резервные копии приватных ключей также могут храниться на бумаге (см [Бумажные кошельки](#)) или на внешнем носителе, таком как флэш-накопитель. Но что, если будет украдена или потеряна сама резервная копия? Эти противоречивые цели безопасности привели к введению портативного и удобного стандарта шифрования закрытых ключей Bitcoin Improvement Proposal 38 или BIP0038 (см [\[bip0038\]](#)).

BIP0038 предлагает общий стандарт для шифрования закрытых ключей при помощи кодовой

фразы и кодирования с помощью Base58Check так, что ключи могут надежно храниться на резервных носителях, передаваться между кошельками, или находиться в каких-либо других условиях, при которых ключ может подвергаться обнаружению. Стандартом для шифрования здесь выбран принятый в National Institute of Standards and Technology (NIST) и широко используемый для коммерческого и военного применения.

А схема шифрования VIP0038 принимает на входе приватный ключ, обычно кодируемый в Wallet Import Format (WIF), в виде строки Base58Check с приставкой «5». Дополнительно, схема шифрования VIP0038 принимает ключевую фразу — длинный пароль, обычно состоящий из нескольких слов или сложную последовательность алфавитно-цифровых символов. На выходе VIP0038 получается зашифрованный приватный ключ в кодировке Base58Check, который начинается с префикса 6P. Если вы видите ключ, который начинается с 6P, это означает, что он зашифрован и для преобразования обратно в формат WIF (префикс 5) потребуется секретная фраза. Многие приложения-кошельки в настоящее время поддерживают VIP0038 и предложат предварительно пользователю ввести секретную фразу для импортирования ключа. Сторонние приложения, такие как невероятно полезное браузерное [Bit Address](#) (Wallet Details tab) могут использоваться для расшифровки ключей VIP0038.

Самый распространенный случай использования VIP0038 — это шифрование ключей для бумажных кошельков, которые могут быть использованы для резервного копирования закрытых ключей на листе бумаги. Если пользователь выбирает сильную секретную фразу, бумажный кошелек с VIP0038-зашифрованными приватными ключами невероятно безопасен и является отличным способом для оффлайн хранения (также известного как "холодный кошелек").

Проверьте зашифрованные ключи в [Пример зашифрованного приватного ключа VIP0038](#) с помощью [bitaddress.org](#) для того, чтобы увидеть, как вы можете получить расшифрованный ключ, введя парольную фразу.

Table 10. Пример зашифрованного приватного ключа VIP0038

Приватный ключ (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2Jpbk eyhfsYB1Jcn
Пароль	MyTestPassphrase
Зашифрованный ключ (VIP0038)	6PRTLH6mWa48xSopbU1cKrVjpKbBZxcLRRCdctL J3z5yxE87MobKoXdTsJ

Pay-to-Script Hash (P2SH) и Multi-Sig адреса

Как мы знаем, традиционные Bitcoin адреса начинаются с цифры "1" и выводятся из публичного ключа, который, в свою очередь, является производным от приватного ключа. Любой может отправить биткойны на "1"-адрес, но средства могут быть потрачены только тем, кто представит соответствующую подпись приватным ключом и хеш публичного ключа.

Bitcoin адреса, начинающиеся с цифры "3" — это pay-to-script hash (P2SH) адреса, иногда

ошибочно называемые мульти-подписными или multisig-адресами. Они объявляют бенефициара транзакции в качестве хеша скрипта, а не владельца публичного ключа. Функция была введена в январе 2012 года с BIP0016 (see [\[bip0016\]](#)), и в настоящее время широко применяется, поскольку она обеспечивает возможность добавлять функциональные возможности к самому адресу. В отличие от транзакций на традиционные "1"-адреса, средства, отправленные на "3"-адреса pay-to-public-key-hash (P2PKH) требуют больше, чем представить хеш одного публичного ключа и подпись одним приватным ключом в качестве доказательства права собственности. Требования обозначены в момент создания адреса, в сценарии, и все входы на этот адрес будут обременены с теми же требованиями.

Адрес хеша pay-to-script получается из сценария транзакции, который определяет, кто может распоряжаться выходами транзакции (более подробно см [\[p2sh\]](#)). Для кодирования адреса хеша pay-to-script используется та же функция двойного хеширования, которая используется при создании Биткоин-адреса, только применяется она к сценарию, а не к публичному ключу:

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

Результирующий "хэш сценария" кодируется при помощи Base58Check с версией префикса 5, что дает закодированный адрес, начинающийся с 3. Например P2SH-адрес 3F6i6kwkevjr7AsAd4te2YB2zZyASEm1HM может быть получен с помощью команд Bitcoin Explorer script-encode, sha256, ripemd160, и base58check-encode (см. [\[libbitcoin\]](#)) следующим образом:

```
$ echo dup hash160 [ 89abcdefabbaabbaabbaabbaabbaabbaabbaabba ] equalverify checksig >
script
$ bx script-encode < script | bx sha256 | bx ripemd160 | bx base58check-encode --version
5
3F6i6kwkevjr7AsAd4te2YB2zZyASEm1HM
```

TIP

P2SH не обязательно такой же, как для стандартной мульти-подписной транзакции. P2SH адрес *чаще всего* представляет сценарий мульти-подписи, но он также может представлять собой сценарий, кодирующий другие типы транзакций.

Многоподписные адреса и P2SH

В настоящее время наиболее распространенной реализация функции P2SH— это сценарий мультиподписного адреса. Как следует из названия, основной сценарий требует более одной подписи для доказательства права собственности и, следовательно, возможности распоряжения средствами. Мультиподпись М-из-N в Биткоин — это такая возможность, когда требуется М подписей (это число также называют "порогом"), от общего числа N ключей, где М равно или меньше N. Например, Боб владелец кафе из [\[ch01_intro_what_is_bitcoin\]](#) может использовать мульти-подписной адрес, требующий одну подпись из двух от ключа, принадлежащего ему и от ключа, принадлежащего его супруге. Это похоже на "общий счет",

как в традиционном банке, которым может распоряжаться любой из супругов с помощью своей личной подписи. Или, например, Гопеш, веб-дизайнер, которому Боб платит за создание веб-сайта, может иметь мультиподписной адрес вида 2-из-3, который гарантирует, что никакие средства не могут быть потрачены, если по крайней мере, двое из его деловых партнеров не подпишут транзакцию.

Мы рассмотрим, как создаются транзакции, которые тратят средства из P2SH (и мультиподписных) адресов в [\[transactions\]](#).

Тщеславные адреса

Адреса тщеславия — это действующие Bitcoin-адреса, содержащие в себе человеческий текст. Например, 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 — это адрес с буквами из алфавита Base-58 на первых четырех позициях, образующими слово «любовь». Тщеславие адреса требуют генерации и проверки миллиардов закрытых ключей в поисках адресов с желаемым шаблоном. Хотя для алгоритма поиска существуют некоторые оптимизации, процесс по существу представляет собой подбор приватного ключа случайным образом, получение публичного ключа и Биткоин-адреса, и проверки на соответствие шаблону повторенный миллиарды раз.

Как только желаемый шаблон для адреса тщеславия найден, приватный ключ, из которого он был получен может быть использован его владельцем для траты биткоинов точно так же, как и любой другой адрес. Адреса тщеславия не менее и не более безопасны, чем любые другие адреса. Они зависят от той же криптографии эллиптических кривых алгоритмов хеширования, что и любые другие адреса. Приватные ключи такого адреса найти не проще, чем для любого другого адреса.

В главе [\[ch01_intro_what_is_bitcoin\]](#) мы познакомили читателя с Евгенией, директором детского благотворительного фонда с Филиппин. Давайте предположим, что Евгения организует сбор средств в Биткоин и хочет использовать тщеславный адрес для популяризации сбора средств. Евгения хочет создать адрес тщеславия, который бы начинался с "1Kids", давайте посмотрим, как создаются подобные тщеславные адреса и что это означает для безопасности мероприятия.

Создание тщеславных адресов

Важно понимать, что Биткоин-адрес — это просто число, представленное в виде символов из алфавита Base58. Поиск по шаблону проде "1Kids" можно рассматривать как поиск адреса в диапазоне от 1Kids111111111111111111111111 до 1KIdsszzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz. В этом диапазоне существует около 58^{29} (примерно $1.4 \cdot 10^{51}$) адресов начинающихся с "1Kids". В таблице Диапазон адресов тщеславия, начинающихся с "1Kids" показаны диапазоны адресов с префиксом 1Kids.

Table 11. Диапазон адресов тщеславия, начинающихся с "1Kids"

[illegible]

может заказать поиск тщеславного адреса с семисимвольной подстрокой и получить результаты в течение нескольких часов вместо того, чтобы искать на собственном процессоре месяцами.

Генерирование адреса тщеславия—это простой перебор: придумать случайный ключ, проверить полученный адрес на соответствие желаемой структуре, повторить, пока не будет найден. В [Переборщик адресов тщеславия](#) показан пример "тщеславного майнера", программы на C++, предназначенной для поиска адресов тщеславия. В примере используется библиотека, которую мы описали в [\[alt_libraries\]](#).

Example 8. Переборщик адресов тщеславия

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

// The string we are searching for
const std::string search = "1kid";

// Generate a random secret key. A random 32 bytes.
bc::ec_secret random_secret(std::default_random_engine& engine);
// Extract the Bitcoin address from an EC secret.
std::string bitcoin_address(const bc::ec_secret& secret);
// Case insensitive comparison with the search string.
bool match_found(const std::string& address);

int main()
{
    // random_device on Linux uses "/dev/urandom"
    // CAUTION: Depending on implementation this RNG may not be secure enough!
    // Do not use vanity keys generated by this example in production
    std::random_device random;
    std::default_random_engine engine(random());

    // Loop continuously...
    while (true)
    {
        // Generate a random secret.
        bc::ec_secret secret = random_secret(engine);
        // Get the address.
        std::string address = bitcoin_address(secret);
        // Does it match our search string? (1kid)
        if (match_found(address))
        {
            // Success!
            std::cout << "Found vanity address! " << address << std::endl;
            std::cout << "Secret: " << bc::encode_hex(secret) << std::endl;
            return 0;
        }
    }
}
```

```

    }
    // Should never reach here!
    return 0;
}

bc::ec_secret random_secret(std::default_random_engine& engine)
{
    // Create new secret...
    bc::ec_secret secret;
    // Iterate through every byte setting a random value...
    for (uint8_t& byte: secret)
        byte = engine() % std::numeric_limits<uint8_t>::max();
    // Return result.
    return secret;
}

std::string bitcoin_address(const bc::ec_secret& secret)
{
    // Convert secret to pubkey...
    bc::ec_point pubkey = bc::secret_to_public_key(secret);
    // Finally create address.
    bc::payment_address payaddr;
    bc::set_public_key(payaddr, pubkey);
    // Return encoded form.
    return payaddr.encoded();
}

bool match_found(const std::string& address)
{
    auto addr_it = address.begin();
    // Loop through the search string comparing it to the lower case
    // character of the supplied address.
    for (auto it = search.begin(); it != search.end(); ++it, ++addr_it)
        if (*it != std::tolower(*addr_it))
            return false;
    // Reached end of search string, so address matches.
    return true;
}

```

NOTE

Пример выше использует `std::random_device`. В зависимости от реализации он может обращаться к криптографически безопасному генератору случайных чисел (КБГСЧ), предоставляемому операционной системой. В случае UNIX-подобной операционной системы, такой как Linux, он запрашивает `/dev/urandom`. В то время как генератор случайных чисел, используемый здесь исключительно в демонстрационных целях, *не* подходит для генерации Биткоин-ключей "производственного качества", так как не обладает с достаточной безопасностью.

Код примера должен быть скомпилирован с использованием компилятора C и слинкован с библиотекой `libbitcoin` (которая должна быть перед этим установлена в системе). Чтобы запустить пример, запустите бинарник `vanity-miner++` без параметров (см [Компилирование и запуск примера перебора адресов тщеславия](#)) и он будет пытаться найти адрес тщеславия, начинающийся с "1kid".

Example 9. Компилирование и запуск примера перебора адресов тщеславия

```
$ # Compile the code with g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Запуск примера
$ ./vanity-miner
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Повторный запуск с другим результатом
$ ./vanity-miner
Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Использование команды "time" для замера времени поиска
$ time ./vanity-miner
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

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

Код примера потребует несколько секунд для того, чтобы найти совпадение трехсимвольного шаблона "kid", в чем мы можем убедиться, если используем Unix-команду `time` для измерения времени выполнения. Поменяем шаблон `search` в исходном коде и посмотрим, сколько времени потребуется для четырех- или пятисимвольных подстрок!

Безопасность тщеславных адресов

Адреса тщеславия могут быть использованы в целях повышения безопасности *и* в обратных

целях; они действительно палка о двух концах. Атакующим сложнее подменить подобный адрес и заставить ваших клиентов платить им вместо вас, с другой стороны, все-таки подделав ваш адрес, обманывать ваших клиентов становится проще.

Евгения может рекламировать случайно сгенерированный адрес (например, 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy), на который люди могут отправлять свои пожертвования. Либо она может сгенерировать тщеславный адрес, который начинается с 1Kids, и сделать его более отличительным.

В обоих случаях, один из рисков, связанных с использованием одного фиксированного адреса (а не выделенного динамического адреса для каждого донора) является то, что вор может быть в состоянии проникнуть на ваш сайт и заменить этот адрес своим собственным, тем самым переведя пожертвования на себя. Если вы рекламировали свой адрес пожертвования в ряде различных мест, пользователи могут визуально проверить адрес до платежа и убедиться, что он является тем же, что и на вашем сайте, в вашей электронной почте или на вашем буклете. В случае случайного адреса, наподобие 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, средний пользователь возможно проверит только несколько первых символов "1J7mdg" и удовлетворится этим совпадением. С помощью генератора адресов тщеславия, кто-то с намерением подмены на аналогичного вида, может быстро сгенерировать адрес, где первые несколько символов будут совпадать, как показано на [Генерируем тщеславный адрес, похожий на случайный адрес](#).

Table 13. Генерируем тщеславный адрес, похожий на случайный адрес

Оригинальный случайный адрес	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Тщеславный (совпадение 4-ех символов)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Тщеславный (совпадение 5-ти символов)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Тщеславный (совпадение 6-ти символов)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Итак, увеличивают ли безопасность адреса тщеславия? Если Евгения создаст адрес тщеславия 1Kids33q44erFfpeXrmDSz7zEqG2FesZEN, пользователи скорее всего заметят шаблон и несколько символов за ним, например "1Kids33". Это заставит злоумышленников, подобрать тщеславный адрес, соответствующий не менее шести символам, затратив в 3364 раз (58×58) больше ресурсов, чем понадобилось Евгении. Таким образом, ресурсы, которые потратит Евгения (или заплатит пулу за поиск подходящего адреса) заставляет злоумышленников искать более длинную подстроку. Если Евгения заплатит пулу за поиски адреса тщеславия длиной 8 символов, то злоумышленнику потребуется адрес длиной 10 символов, что неосуществимо на персональном компьютере и дорого даже с использованием специального оборудования или пула. То, что доступно Евгении становится недоступным для атакующего, особенно если потенциальное вознаграждение от мошенничества недостаточно высоко, чтобы покрыть стоимость генерации адреса тщеславия.

Бумажные кошельки

Бумажные Bitcoin-кошельки — это приватные ключи, напечатанные на бумаге. Часто бумажный кошелек также включает соответствующий Bitcoin-адрес для удобства, но это не является необходимостью, поскольку он может быть получен из приватного ключа. Бумажные кошельки очень эффективный способ резервного копирования или офлайн-хранения, также известного как "холодное хранилище". В качестве резервной копии, бумажный кошелек помогает обезопаситься от потери ключа в результате компьютерного сбоя, такого как повреждение содержимого жесткого диска, кражи или случайного удаления. В качестве механизма "холодного хранения", если ключи бумажного кошелька генерируются офлайн и не хранятся в компьютерной системе, они гораздо более устойчивы от атак хакеров, клавиатурных перехватчиков и других онлайн-компьютерных угроз.

Бумажные кошельки бывают разных форм, размеров, и конструкций, но в основе своей это просто ключ и адрес, напечатанные на бумаге. В таблице [Самый простой вид бумажного кошелька — это распечатка Биткоин-адреса и приватного ключа](#), показан простейший бумажный кошелек.

Table 14. Самый простой вид бумажного кошелька — это распечатка Биткоин-адреса и приватного ключа.

Публичный адрес	Приватный ключ (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2Jpbnk eyhfsYB1Jcn

Бумажные кошельки могут быть легко созданы с помощью такого инструмента, как [bitaddress.org](#), который написан на языке JavaScript и исполняется на стороне клиента. Этот сайт содержит весь код, необходимый для генерации ключей и бумажных кошельков, даже при полном отключении от Интернета. Для того, чтобы использовать его, сохраните HTML-страницу на локальном диске или на внешнем флэш-накопителе. Отключитесь от Интернета и откройте HTML-файл в браузере. Будет еще лучше, если вы перезагрузитесь с "чистой" операционной системой, например с загрузочного CD-ROM с Linux. Любые ключи, сгенерированные в офлайне с помощью этого инструмента, можно распечатать на локальном принтере, подключенном через USB-кабель (не на беспроводном), тем самым создавая бумажные кошельки, ключи от которых существуют только на бумаге и никогда не были сохранены в какой-нибудь онлайн-системе. Поместите эти бумажные кошельки в неогороженный шкаф и пошлите необходимую сумму биткоинов на адреса этих бумажных кошельков. Таким образом вы получили простое, но очень эффективное решение для "холодного хранения". На [Пример простого бумажного кошелька с сайта bitaddress.org](#) показан бумажный кошелек созданный с помощью сайта [bitaddress.org](#).



Figure 14. Пример простого бумажного кошелька с сайта bitaddress.org

Недостатком простых бумажных кошельков является их слабая физическая безопасность. Вор, имеющий физический доступ может либо украсть либо сфотографировать ключи и взять под свой контроль соответствующие адреса. Более сложная система хранения бумажного кошелька использует зашифрованные приватные ключи VIP0038. Ключи, напечатанные на бумаге защищены ключевой фразой, запомненной владельцем. Без ключевой фразы, зашифрованные ключи бесполезны. Подобные кошельки много лучше обычных запаролированных кошельков, т.к. эти ключи никогда не были онлайн и должны быть физически извлечены из сейфа или другого физически защищенного хранилища. В [Пример зашифрованного бумажного кошелька с сайта bitaddress.org. Пароль "test"](#). показан бумажный кошелек с VIP0038-зашифрованным секретным ключом, созданным с помощью сайта bitaddress.org.



Figure 15. Пример зашифрованного бумажного кошелька с сайта bitaddress.org. Пароль "test".

WARNING

Хотя вы можете внести деньги в бумажный кошелек несколько раз, выводить из них средства следует за один раз. Это потому, что в процессе разблокирования и расходования средств некоторые кошельки могут генерировать адреса сдачи, если вы тратите меньше, чем весь объем. Кроме того, если используемый вами компьютер скомпрометирован, вы рискуете потерять секретный ключ. Потратив весь баланс бумажного кошелька только один раз, вы уменьшаете риск компрометации ключа. Если вам нужно только небольшое количество, отправьте оставшиеся средства на новый бумажный кошелек в той же транзакции.

Бумажные кошельки бывают различных дизайнов и размеров. Некоторые из них предназначены для вручения в качестве подарков и украшены соответственно, например, в виде открытки на Рождество или Новый год. Другие предназначены для хранения в банковской ячейке или сейфе, с приватным ключом, скрытым в некотором роде либо непрозрачными наклейками, которые нужно сдирать, либо сложенные и запечатанные защищенной от подделки клейкой фольгой. На рисунках [through](#) показаны различные варианты бумажных кошельков для безопасного хранения и резервного копирования.

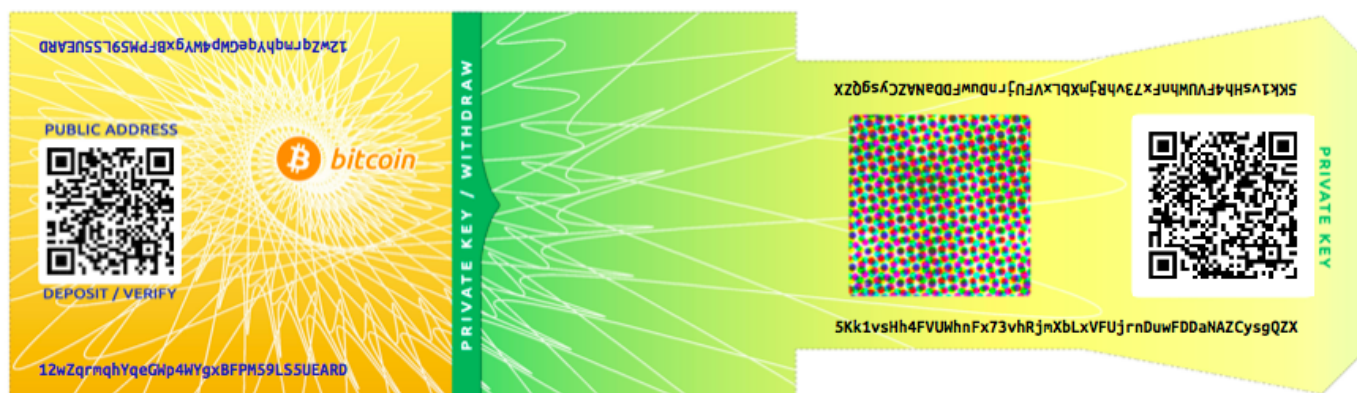


Figure 16. Пример бумажного кошелька с сайта bitcoinpaperwallet.com с приватным ключом на сгибаемом вкладыше.



Figure 17. Бумажный кошелек с сайта bitcoinpaperwallet.com со скрытым приватным ключом.

Другие конструкции содержат дополнительные копии ключа и адреса в виде съемных заглушек, похожих на корешки билетов, что позволяет хранить несколько копий для защиты от пожара, наводнения или других стихийных бедствий.



Figure 18. Пример бумажног кошелька с дополнительными копиями ключей на резервной "заглушке".

Транзакции

Введение

Транзакции являются наиболее важной частью системы Биткоин. Все остальное в Биткоин предназначено для обеспечения нормальных операций по созданию, распространению по сети, проверки, и, наконец, добавления к глобальной книге сделок (blockchain). Транзакции — это структуры данных, которые кодируют передачу ценности между участниками системы. Каждая транзакция соответствует записи в глобальной бухгалтерской книге под названием blockchain.

В этой главе мы рассмотрим все различные формы транзакций, что они содержат, как их создавать, как они проверяются и как попадают в постоянную базу всех операций.

Цикл жизни транзакции

Жизненный цикл транзакции начинается с ее создания, также известного как *возникновения*. Транзакция затем подписывается одним или несколькими подписями, означающими разрешение перевода средств, на которые ссылается транзакция. Далее транзакция транслируется по сети Биткоин, где каждый узел сети (участник) проверяет и передает дальше транзакцию по сети до тех пор, пока та не достигнет (почти) каждого узла в сети. Наконец, транзакция проверяется узлом майнера и включается в блок вместе с остальными попадая в blockchain.

После попадания в blockchain и подтверждения достаточным количеством последующих блоков, транзакция становится неотъемлемой частью бухгалтерской книги и признается действительной всеми участниками. Средства, полученные новым владельцем при помощи транзакции, могут быть потрачены при помощи новой транзакции, удлиняя цепь владения и начиная жизненный цикл транзакции снова.

Создание транзакций

В некотором смысле полезно думать о транзакции, как о бумажном чеке. Как и чек, транзакция является инструментом, который выражает намерение перевести деньги и для финансовой системы он невидим до тех пор, пока не будет заявлен для исполнения. Как и в случае чека, инициатором транзакции не должно быть то же лицо, что и подписант.

Транзакции могут создаваться онлайн или офлайн кем угодно, даже если лицо, создавшее транзакцию не является уполномоченным подписывать распоряжения по счету. Например, управляющий счетами клерк может обрабатывать чеки на подпись генеральным директором. Аналогичным образом, тот же самый управляющий может создавать транзакции Биткоин, на которые затем генеральный директор поставит свои цифровые подписи. В то время как чек ссылается на конкретный счет в качестве источника средств, Биткоин-транзакция ссылается на конкретную предыдущую транзакцию в качестве источника, а не счет.

После того, как транзакция была создана, она подписывается владельцем (или владельцами) источника средств. Если она правильно сформирована и подписана, транзакция становится действительной и содержит всю информацию, необходимую для выполнения перевода денежных средств. И, наконец, действительная транзакция должна достичь сети Биткоин так, чтобы она имела возможность распространиться и достигнуть майнерского узла, который включит ее в блокчейн.

Распространение транзакций по Биткоин-сети

Первым делом транзакция должна быть транслирована в Биткоин-сеть для того, чтобы она могла продолжить распространение и быть включена в blockchain. В сущности, Биткоин-транзакция имеет размер всего лишь от 300 до 400 байтов и должна достичь любого из десятков тысяч Биткоин-узлов. Отправители не обязаны доверять узлам, которые они используют для передачи транзакции, до тех пор, пока они используют более одного. Узлы не должны доверять отправителю или установить "личность" отправителя. Поскольку транзакция будет подписана и не содержит конфиденциальную информацию, приватные ключи, или учетные данные, она может быть публично транслирована с помощью любого удобного сетевого транспорта. В отличие от, например, сделок по кредитным картам, которые содержат конфиденциальную информацию и могут быть переданы только по зашифрованным сетям, Биткоин-транзакция может быть отправлена с помощью любой сети. Не имеет значения каким образом транзакция достигнет первого Биткоин-узла, который сможет распространить ее дальше в сеть.

Отсюда, Биткоин-транзакции могут быть переданы в Биткоин-сеть по незащищенным каналам, таким, как Wi-Fi, Bluetooth, NFC, Chirp, штрих-коды, или путем копирования и вставки в веб-формы. В крайних случаях, Биткоин-транзакции могут быть переданы посредством пакетной спутниковой или коротковолновой радиосвязи и тд. Биткоин-транзакция может даже быть закодирована в виде смайликов и размещена на публичном форуме или отправлена при помощи SMS или по скайпу. Биткоин превратил деньги в структуру данных, так что стало практически невозможным запретить кому-либо создавать и выполнять Биткоин-транзакции.

Распространение транзакций по Биткоин-сети

Как только транзакция отправляется к любому узлу, подключенному к Биткоин-сети, она подтверждается этим узлом. Если она окажется действительной, этот узел ретранслирует ее на другие узлы, к которым он сам подключен, а отправитель синхронно получит сообщение об успехе. Если транзакция окажется недействительной, узел отклонит ее и синхронно сообщит об отклонении отправителю.

Сеть Биткоин является пиринговой, что означает, что каждый Биткоин-узел связан с несколькими другими узлами, которые он обнаружил сразу после запуска при помощи децентрализованного протокола. Вся сеть образует слабо связанную сетку без фиксированной топологии или какой-либо структуры, что делает ее одноранговой, а узлы равными. Сообщения, в том числе транзакции и блоки, которые каждый узел ретранслирует узлам, к которым сам подключен — это процесс, называемый "наводнением". Новая подтвержденная

транзакция, полученная любым узлом сети, будет отправлена всем узлам, подключенным к нему (соседям), каждый из которых отправит транзакцию в свою очередь уже всем своим соседям, и так далее. Таким образом, в течение нескольких секунд действительная транзакция распространится в виде волны, расширяющейся с геометрической прогрессией, пока ее не получат все узлы сети.

Сеть Биткоин предназначена для эффективного и надежного распространения транзакций и блоков всем своим узлам. Для предотвращения спама, атак на отказ в обслуживании, или отражения других нападков на систему, каждый узел независимо проверяет каждую транзакцию, прежде чем передать ее дальше. Искорженная сделка не будет передана далее одного узла. Правила, по которым проверяются транзакции объясняются более подробно в [\[tx_verification\]](#).

Структура транзакции

Транзакция — это *структура данных*, которая кодирует передачу стоимости из источника средств, называемого *входом*, получателю, называемому *выходом*. Входы и выходы транзакции не связаны со счетами или владельцами. Вместо этого, вы должны думать о них, просто как о количествах, суммах, запертых при помощи определенного ключа так, что только владелец или человек, владеющий ключом, может их разблокировать. Транзакция содержит некоторые поля, как это показано на [Структура транзакции](#).

Table 1. Структура транзакции

Размер	Поле	Описание
4 байта	Версия	Описывает каким правилам подчиняется данная транзакция
1-9 байтов (VarInt)	Счетчик входов	Число входов транзакции
Переменный	Входы	Один или несколько входов транзакции
1-9 байтов (VarInt)	Счетчик выходов	Число выходов транзакции
Переменный	Выходы	Один или несколько выходов транзакции
4 байта	Locktime	UNIX-время или номер блока

Срок связывания транзакции

Срок связывания, также известный как `nLockTime` по имени переменной, используемой в базовом клиенте, определяет самое раннее время, когда транзакция может стать действительной и может быть передана в сеть или добавлена в `blockchain`. В большинстве транзакций он установлен в нуль, что означает немедленное распространение и исполнение. Если поле отлично от нуля и меньше 500 млн, то оно интерпретируется, как высота блока, то есть сделка не является действительной, не передается и не включается в `blockchain` до наступления заданной высоты блока. Если оно больше 500 миллионов, то интерпретируется как Unix-время (количество секунд, прошедших с 1 января 1970-го года, и транзакция не может быть признана действительной до указанного времени. Транзакции с указанием срока связывания должны быть сохранены их создателем и переданы в сеть Биткоин только после того, как они вступят в силу. Использование срока связывания эквивалентно постдатированию бумажного чека.

Входы и выходы транзакции

Главный строительный блок Биткоин-транзакции — это *непотраченный выход транзакции* (unspent transaction output) или UTXO. UTXO — это неделимые куски биткоинов, привязанные к конкретному владельцу, записанные в `blockchain`, и признанные валютой во всей сети. Сеть Bitcoin отслеживает все доступные (неизрасходованные) UTXO на данный момент насчитывающие миллионы. Всякий раз, когда пользователь получает Биткоин, эта сумма записывается в `blockchain` как UTXO. Таким образом, биткоины пользователя могут быть разбросаны по UTXO среди сотен транзакций и сотен блоков. В качестве следствия, не существует такого понятия, как баланс Биткоин-адреса или счета; есть только отдельные UTXO, привязанные к конкретным владельцам. Понятие баланса — это конструкция, которой оперирует приложение кошелька. Кошелек вычисляет баланс пользователя путем сканирования блокчейна и агрегирования все UTXO принадлежащие этому пользователю.

TIP

В Биткоин не существует понятия счетов или балансов; Есть только *непотраченные выходы транзакций* (UTXO), разбросанные по `blockchain`.

UTXO может содержать произвольное значение кратное сатоши. Так же, как значения в долларах могут содержать два знака после запятой в виде центов, биткоины можно дробить до восьми знаков после запятой на сатоши. Хотя UTXO может содержать любое произвольное значение, после своего создания он неделим, как монета, которую нельзя разрезать пополам. Если в UTXO больше требуемого значения транзакции, он все равно должен быть потрачен полностью, а разница должна оказаться в виде сдачи транзакции. Другими словами, если у вас есть 20 биткоинов в UTXO и вы хотите заплатить 1 биткоин, ваша транзакция должна потребить все 20 биткоинов UTXO и создать два выхода: один с переводом 1 биткоина получателю, а другой с переводом 19 биткоинов в виде сдачи обратно на ваш кошелек.

Представьте себе покупателя напитка стоимостью \$1.50, который роется в кошельке и пытается найти комбинацию монет и банкнот, составляющих \$1.50. Покупатель постарается использовать точные деньги (доллар и две монеты по 25 центов), или комбинацию небольших номиналов (шесть 25-ти центовых монет), или, если не получается, использует банкноту с большим номиналом. Заплатив, скажем, \$5, покупатель ожидает, что владелец магазина, выдаст \$3.50 в виде сдачи, которая вернется в кошелек и станет возможна для будущих сделок.

Аналогичным образом, Bitcoin-транзакция должна быть создана из UTXO пользователя в любом доступном номинале. Пользователи не могут порезать UTXO пополам так же, как нельзя порезать пополам долларовую купюру и использовать эти кусочки в качестве денег. Приложение кошелька пользователя, как правило, выбирает из имеющихся различных UTXO и составляет сумму, превышающую или равную желаемой сумме транзакции.

Как и в реальной жизни, приложение может использовать несколько стратегий получения требуемой суммы: объединить несколько мелких ячеек до точного значения, или, использовать большую сумму и рассчитывать на сдачу. Все эти сложные расчеты производятся автоматически ПО кошелька пользователя. Управлять этим имеет смысл только, если вы программно собираете сырые транзакции из UTXO.

UTXO, потребляемые транзакциями называются входами транзакций, а UTXO, создаваемые транзакциями называются выходами транзакций. Таким образом биткойны перемещаются от владельца к владельцу по цепочке транзакций, потребляющих и создающих UTXO. Транзакции потребляют UTXO, отпирая их подписью нынешнего владельца и создают UTXO, запирая их Биткойн-адресом нового владельца.

Исключение выходной и входной цепи представляет собой особый тип сделки называется *coinbase* сделка, которая является первой сделкой, в каждом блоке. Эта сделка находится там на "выигрышный" горняка и создает совершенно новый Bitcoin, подлежащего уплате в этом шахтера в качестве награды за добычи полезных ископаемых. Это, как создается предложение денег Bitcoin во время процесса добычи, как мы увидим в <ch8> >.

ТИП

Что сначала? Входы или выходы, курица или яйцо? Строго говоря, выходы в идут первыми потому, что *coinbase*-транзакции, создающие новые монеты, не имеют входов и создают выходы из ничего.

Выходы транзакции

Каждая Bitcoin-транзакция создает выходы, которые записываются в бухгалтерскую книгу Биткойн. Почти все эти выходы, за одним исключением (см [Вывод данных \(OP_RETURN\)](#)) составляют *непотраченные выходы транзакций* или UTXO (*unspent transaction outputs*), которые затем распознаются всей сетью и доступны владельцу для использования в будущих транзакциях. Отправка кому-то биткойнов — это создание неизрасходованного выхода транзакции (UTXO), зарегистрированного на адресата.

UTXO отслеживаются каждым полным узлом Bitcoin в виде набора данных называемого *множество UTXO* или *пулом UTXO*, содержащегося в базе данных. Новые транзакции

потребляют (расходуют) один или несколько из этих выходов из множества UTXO.

Выходы транзакции состоят из двух частей:

- Сумма в биткоинах, номинированных в *сатошах*, наименьшей расчетной единицы в Биткоин
- *запирающий сценарий*, также известный как "обременение", которое "запирает" эту сумму, указав условия, которые должны быть соблюдены, чтобы можно было потратить выходы.

Язык сценариев транзакций, использованным в запирающем сценарии упомянутый ранее, подробно обсуждается в [Сценарии транзакций и язык сценариев](#). Структура выхода транзакции показывает структуру выхода транзакции.

Table 2. Структура выхода транзакции

Размер	Поле	Описание
8 байтов	Сумма	Количество в сатошах (10^{-8} bitcoin)
1-9 байтов (VarInt)	Размер запирающего сценария	Длина в байтах
Переменная	Запирающий сценарий	Сценарий, определяющий условия, удовлетворение которых требуется для того, чтобы можно было потратить выход

В [Сценарий, который вызывает API blockchain.info, чтобы найти UTXO, связанную с адресом](#) мы используем API сайта blockchain.info для того, чтобы найти неиспользованные выходы (UTXO) определенного адреса.

Example 1. Сценарий, который вызывает API `blockchain.info`, чтобы найти UTXO, связанную с адресом

```
# get unspent outputs from blockchain API

import json
import requests

# example address
address = '1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX'

# The API URL is https://blockchain.info/unspent?active=<address>
# It returns a JSON object with a list "unspent_outputs", containing UTXO, like this:
#{  "unspent_outputs":[
#    {
#      "tx_hash":"ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167",
#      "tx_index":51919767,
#      "tx_output_n": 1,
#      "script":"76a9148c7e252f8d64b0b6e313985915110fcfefcf4a2d88ac",
#      "value": 8000000,
#      "value_hex": "7a1200",
#      "confirmations":28691
#    },
#    ...
#  ]}

resp = requests.get('https://blockchain.info/unspent?active=%s' % address)
utxo_set = json.loads(resp.text)["unspent_outputs"]

for utxo in utxo_set:
    print "%s:%d - %ld Satoshis" % (utxo['tx_hash'], utxo['tx_output_n'],
    utxo['value'])
```

Запустив скрипт, мы видим список идентификаторов транзакций, двоеточие, порядковый номер конкретного неизрасходованного выхода транзакции (UTXO), и стоимость этого UTXO в сатошах. Запирающий сценарий не показан в выходе в [Запускаем сценарий `get-utxo.py`](#).

```
$ python get-utxo.py
ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167:1 - 8000000 Satoshi
6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 - 16050000
Satoshi
74d788804e2aae10891d72753d1520da1206e6f4f20481cc1555b7f2cb44aca0:0 - 5000000 Satoshi
b2affea89ff82557c60d635a2a3137b8f88f12ec85082f7d0a1f82ee203ac4:0 - 10000000
Satoshi
...
```

Условия траты (обременения)

Выходы транзакций ассоциируют определенную сумму (в сатоши) и определенное *обременение* или запирающий сценарий, который определяет условия, возволяющие вывод средств. В большинстве случаев, запирающий скрипт блокирует выход на определенный адрес Bitcoin, тем самым передавая право собственности на эту сумму новому владельцу. Когда Алиса заплатила Бобу за чашку кофе, ее транзакция создала выход размером 0.015 биткоинов с *обременением* запирающим средства на адрес кафе. Это выход в 0.015 биткоинов был записан в blockchain и стал частью множества *Неизрасходованных Выходов Транзакций*, то есть он показался в бумажнике Боба в виде части имеющегося остатка. Когда Боб решит потратить эту сумму, его транзакция освободит обременение и откроет выход, предоставив сценарий отпирания, содержащий подпись секретного ключа Боба.

Входы транзакции

Проще говоря, входы транзакций являются указателями на UTXO. Они ссылаются на конкретную UTXO с помощью хэша транзакции и порядкового номера. Для распоряжения UTXO, вход транзакции также включает в себя отпирющие сценарии, которые удовлетворяют условиям расходования, установленных UTXO. Отпирющий сценарий, как правило — это подпись, доказывающая право собственности на адрес Bitcoin, который находится в запирающем сценарии.

Когда пользователи совершают платеж, их кошелек собирает транзакцию путем выбора из имеющихся UTXO. Например, чтобы сделать платеж 0.015 биткоинов, приложение кошелька может выбрать UTXO с 0.01 и UTXO с 0.005, сложив их для получения требуемой суммы.

В [Сценарий для расчета общего количества биткоинов will be issued](#) показано использование "жадного" алгоритма выбора определенной суммы из доступных UTXO. В примере, доступный UTXO представлены в виде константного массива, но в реальности, доступные UTXO могут быть получены с помощью RPC-вызова Bitcoin Core или сторонних API, как показано в [Сценарий, который вызывает API blockchain.info, чтобы найти UTXO, связанную с адресом](#).

```

# Selects outputs from a UTXO list using a greedy algorithm.

from sys import argv

class OutputInfo:

    def __init__(self, tx_hash, tx_index, value):
        self.tx_hash = tx_hash
        self.tx_index = tx_index
        self.value = value

    def __repr__(self):
        return "<%s:%s with %s Satoshis>" % (self.tx_hash, self.tx_index,
                                              self.value)

# Select optimal outputs for a send from unspent outputs list.
# Returns output list and remaining change to be sent to
# a change address.
def select_outputs_greedy(unspent, min_value):
    # Fail if empty.
    if not unspent:
        return None
    # Partition into 2 lists.
    lessers = [utxo for utxo in unspent if utxo.value < min_value]
    greater = [utxo for utxo in unspent if utxo.value >= min_value]
    key_func = lambda utxo: utxo.value
    if greater:
        # Not-empty. Find the smallest greater.
        min_greater = min(greater)
        change = min_greater.value - min_value
        return [min_greater], change
    # Not found in greater. Try several lessers instead.
    # Rearrange them from biggest to smallest. We want to use the least
    # amount of inputs as possible.
    lessers.sort(key=key_func, reverse=True)
    result = []
    accum = 0
    for utxo in lessers:
        result.append(utxo)
        accum += utxo.value
        if accum >= min_value:
            change = accum - min_value
            return result, "Change: %d Satoshis" % change
    # No results found.
    return None, 0

def main():
    unspent = [

```

```

OutputInfo("ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167", 1,
8000000),

OutputInfo("6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf", 0,
16050000),

OutputInfo("b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac4", 0,
10000000),

OutputInfo("7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1", 0,
25000000),

OutputInfo("55ea01bd7e9afd3d3ab9790199e777d62a0709cf0725e80a7350fdb22d7b8ec6", 17,
5470541),

OutputInfo("12b6a7934c1df821945ee9ee3b3326d07ca7a65fd6416ea44ce8c3db0c078c64", 0,
10000000),

OutputInfo("7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818", 0,
16100000),
    ]

    if len(argv) > 1:
        target = long(argv[1])
    else:
        target = 55000000

    print "For transaction amount %d Satoshis (%f bitcoin) use: " % (target,
target/10.0**8)
    print select_outputs_greedy(unspent, target)

if __name__ == "__main__":
    main()

```

Если запустить сценарий *select-utxo.py* без параметров, он будет пытаться построить множество UTXO (и сдачу) для платежа 55,000,000 сатоши (0.55 биткоинов). Если в параметре передать требуемую сумму платежа, скрипт выберет соответствующие UTXO для этой цели. В [Запускаем сценарий select-utxo.py](#) показан пример запуска скрипта с платежом 0.5 биткоина или 50,000,000 сатоши.

Example 4. Запускаем сценарий `select-utxo.py`

```
$ python select-utxo.py 50000000
For transaction amount 50000000 Satoshis (0.500000 bitcoin) use:
([<7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1:0 with 25000000
Satoshis>, <7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818:0 with
16100000 Satoshis>,
<6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 with 16050000
Satoshis>], 'Change: 7150000 Satoshis')
```

После того, как UTXO выбран, кошелек генерирует отпирющие скрипты, содержащие подписи для каждого из UTXO, тем самым делая их расходными удовлетворяя условиям запирающего сценария. Кошелек добавляет эти UTXO и отпирющие скрипты в качестве входов транзакции. В [Структура входа транзакции](#) показана структура входа транзакции.

Table 3. Структура входа транзакции

Размер	Поле	Описание
32 байта	Хэш транзакции	Указатель на транзакцию, содержащую UTXO, который будет потрачен
4 байта	Номер выхода	Порядковый номер UTXO; начиная с 0
1-9 байтов (VarInt)	Размер отпирющего скрипта	Длина скрипта в байтах
Переменная	Отпирющий сценарий	Сценарий, который выполняет условия запирающего UTXO сценария.
4 байта	Номер последовательности	В настоящее время отключенная функция замены транзакции, установлен в 0xFFFFFFFF

NOTE

Порядковый номер используется для переопределения транзакции до истечения locktime, возможности, которая в настоящее время отключена в Bitcoin. В большинстве транзакций это значение установлено в максимальное значение (0xFFFFFFFF) и игнорируется сетью Bitcoin. Если в транзакции установлен ненулевой locktime, по меньшей мере, один из ее входов должен иметь порядковый номер меньше 0xFFFFFFFF.

Комиссионные транзакции

Большинство транзакций включают в себя комиссионные, которые идут майнерам в качестве

компенсации за защиту сети Биткоин. Майнинг, комиссионные и награды майнеров обсуждаются более подробно в [\[ch8\]](#). В этом разделе рассматривается, как комиссионные включаются в типичную транзакцию. Большинство кошельков рассчитывает и включают комиссионные автоматически. Тем не менее, если вы создаете транзакции программно или с помощью интерфейса командной строки, вы должны сами рассчитывать и включать эти комиссионные.

Комиссионные транзакций служат стимулом для включения транзакции в следующий блок, а также в качестве обратного стимула для «спамных» транзакций или каких-либо злоупотреблений в системе, путем навязывания небольшой стоимости каждой транзакции. Комиссионные получает майнер нашедший блок и записавший транзакцию в блокчейн.

Комиссионные за транзакцию рассчитываются на основе размера транзакции в килобайтах, а не значения переводимой суммы. В целом, комиссионные устанавливаются на основе рыночных сил внутри сети Bitcoin. Майнеры приоритезируют транзакции на основе различных критериев, включая размер комиссионных, и даже могут обрабатывать транзакции бесплатно при определенных обстоятельствах. Суммы комиссионных влияют на приоритет обработки, а это означает, что транзакция с достаточными комиссионными, вероятно, будет включена в следующий блок, в то время как транзакции с недостаточными комиссионными или вообще без них может быть задержана, обработана последней или не обработана вообще. Комиссионные не являются обязательными, и транзакции без комиссионных вообще могут быть обработаны в конце концов; тем не менее, включение комиссионных мотивирует приоритетную обработку.

Со временем, способ, которым рассчитываются комиссионные и эффект, который они оказывают на приоритеты, эволюционировал. Сначала комиссионные были зафиксированы и постоянны по всей сети. Постепенно структура комиссионных была изменена таким образом, чтобы появились рыночные силы и регулирование цены на основе пропускной способности сети и объема транзакций сети. В настоящее время минимальный объем комиссионных зафиксирован на уровне 0.0001 Bitcoin или десятой части милли-биткоина за килобайт, снизившись с одного миллибиткоина. Большинство транзакций составляет менее одного килобайта; тем не менее, транзакции с несколькими входами и выходами могут оказаться больше. В будущих версиях протокола Bitcoin, ожидается, что приложения кошельков смогут использовать статистический анализ для расчета наиболее соответствующей в данный момент платы за транзакцию на основании среднего от последних транзакций.

Текущий алгоритм, используемый майнерами для определения приоритета транзакций на попадание в блок на основании размера комиссионных подробно рассмотрен в [\[ch8\]](#).

Добавление комиссионных к транзакции

Структура данных транзакций не содержит поле для комиссионных. Вместо этого, комиссионные подразумеваются как разность между суммой входов и суммой выходов. Любое избыточное количество, остающееся после того, как все выходы были вычтены из всех входов, считается комиссией собираемой майнерами.

Комиссионные транзакций рассчитываются, как разница входов и выходов:

$$\text{Комиссионные} = \text{SUM}(\text{входы}) - \text{SUM}(\text{выходы})$$

Это очень важный момент в транзакциях, который необходимо понимать потому, что если вы создаете свои собственные транзакции, вам необходимо убедиться, что вы случайно не включите очень большие комиссионные. Это означает, что вы должны учитывать абсолютно все входы, и в случае необходимости, указав сдачи, или может так случиться, что в конечном итоге вы дадите майнерам очень большие чаевые!

Например, если вы используете 20-биткойновый UTXO для платежа в 1 биткойн, то вам придется включить в платеж выход для сдачи 19 биткойнов обратно на ваш кошелек. В противном случае, 19-биткойновый "остаток" сочтется комиссией за транзакцию и будет присвоен майнером, нашедшим блок для вашей транзакции. Хотя вы будете обслужены вне очереди и сделаете майнера очень счастливым, это, вероятно, не то, чего вы хотели.

WARNING

Если вы забудете добавить выход для сдачи в построенную вручную транзакцию, вы заплатите сдачи в качестве комиссионных. "Сдачи не надо!" может не входить в ваши планы.

Давайте посмотрим, как это работает на практике, еще раз, посмотрев на покупку кофе Алисой. Алиса хочет потратить 0.015 биткойнов и заплатить за кофе. Для того, чтобы обеспечить быстрое прохождение этой транзакции, она захочет прибавить комиссионные, скажем, 0.001. Это будет означать, что общая стоимость транзакции составит 0.016. Поэтому ее кошелек должен найти источник набора UTXO, в сумме дающий 0.016 биткойнов или более, и, при необходимости включая сдачи. Скажем, ее кошелек содержит 0.2 биткойна доступных в UTXO. Следовательно, этот UTXO можно использовать создав один выход на кафе Боба суммой 0.015, а второй выход с 0.184 биткойнов в виде сдачи обратно в ее собственный кошелек, оставив 0.001 биткойнов в неявном виде в качестве комиссионных за транзакцию.

Теперь давайте посмотрим на другой сценарий. Евгения, наш директор детского благотворительного фонда с Филиппин, завершила сбор средств на приобретение школьных учебников для детей. Она получила несколько тысяч небольших пожертвований от людей по всему миру, на общую сумму 50 биткойнов, так что ее кошелек полон очень мелких платежей (UTXO). Теперь она хочет купить сотни школьных учебников у местного издателя за эти биткойны.

Так как приложение-кошелек Евгении пытается построить одну большую транзакцию, оно должно собрать из имеющегося набора UTXO небольших сум. Это означает, что результирующая транзакция будет иметь более сотни маленьких UTXO источников на входах, и только один выход на счет книжного издательства. Транзакция с таким количеством входов будет иметь размер больше, чем один килобайт, возможно от 2 до 3 килобайт. В результате, это потребует более высокую пошлину, чем минимальная пошлина 0.0001 Bitcoin.

Приложение-кошелек Евгении рассчитает соответствующие комиссионные умножив размер

транзакции на комиссию за килобайт. Многие кошельки переплачивают комиссионные крупных транзакций для обеспечения обработки транзакции в кратчайшие сроки. Комиссионные увеличены не потому, что Евгений переводит больше денег, а потому, что ее транзакция более сложная и больше по размеру—комиссионные в Биткоин не зависят от переводимой суммы.

Сцепление транзакций и осиротевшие транзакции

Как мы уже видели, транзакции образуют цепь, в результате чего одна транзакция тратит выходы предыдущей транзакции (известной в качестве прародителя) и создает выходы для последующей транзакции (известный, как потомок). Иногда целая цепь транзакций в зависимости друг от друга, скажем, прародитель, потомок, и внучатые транзакции создаются одновременно для того, чтобы исполнить сложную транзакционную схему, в которой требуется, чтобы валидные потомки были подписаны до подписания родителя. Например, это метод, используется в CoinJoin транзакциях, где несколько сторон объединяют вместе транзакции, в целях защиты конфиденциальности.

Когда цепочка транзакций передается по сети, они не всегда поступают в том же порядке. Иногда дочерняя транзакция может прибыть раньше родительской. В этом случае узлы, получившие дочернюю видят, что она ссылается на пока неизвестную родительскую транзакцию. Вместо того, чтобы отбросить подобную транзакцию, они помещают ее во временный пул, где она ждет прибытия родительской транзакции и распространяют ее дальше другим узлам. Пул транзакций без родителей известен как *пул сиротских транзакций*. После того, как поступает родительская транзакция, любые сироты, которые ссылаются на UTXO созданный родителем освобождаются из пула, рекурсивно перевалидируются, а затем вся цепочка транзакций может быть включена в пул транзакций, которые уже могут быть помещены в блок. Цепи транзакций могут быть сколь угодно длинными, с любым числом поколений, передаваемых одновременно. Сиротский пул гарантирует, что валидные транзакции не будут отброшены только потому, что их родитель задерживается и что в конечном итоге цепь, которой они принадлежат, реконструируется в правильном порядке, независимо от порядка прибытия.

На количество бесхозных транзакций, хранимых в памяти, имеется лимит, для предотвращения атак на отказ в обслуживании. Этот лимит определяется как MAX_ORPHAN_TRANSACTIONS в исходном коде базового Bitcoin-клиента. Если число бесхозных транзакций в пуле превышает MAX_ORPHAN_TRANSACTIONS, из пула случайным образом начинают удаляться лишние до тех пор, пока размер пула не вернется в допустимые пределы.

Сценарии транзакций и язык сценариев

Биткоин-клиенты валидируют транзакции, выполняя скрипт, написанный на Forth-подобном скриптовом языке. И запирающий скрипт (обременение) на UTXO и разблокирующий скрипт, как правило содержащий подпись, написаны на этом языке сценариев. Когда транзакция подтверждается, выполняется сценарий разблокировки на каждом входе наряду с

соответствующим запирающим сценарием и смотрится удовлетворяется ли условие траты.

Сегодня, большинство транзакций, проходящих через сеть Биткоин имеют вид "Алиса платит Бобу" и основаны на одном и том же сценарии под названием Платеж-по-Хешу-Публичного-Ключа (Pay-to-Public-Key-Hash). Тем не менее, использование скриптов для запираения выходов и разблокирования входов означает, что благодаря использованию языка программирования, транзакции могут содержать бесконечное число условий. Биткоин-транзакции не ограничиваются формой "Алиса платит Бобу".

Это только верхушка айсберга возможностей, которые могут быть выражены при помощи этого языка сценариев. В этом разделе мы покажем компоненты скриптового языка Биткоин-транзакций и покажем, как он может быть использован для выражения сложных условий траты и как эти условия могут быть удовлетворены скриптами разблокировки.

ТИП

Проверка транзакций в Биткоин не основана на статической модели, но вместо этого достигается посредством исполнения скриптового языка. Этот язык позволяет выразить почти бесконечное разнообразие условий. Благодаря ему Биткоин становится "программируемыми деньгами."

Создание сценария (блокировка + разблокировка)

Механизм проверки Биткоин-транзакций опирается на два типа сценариев: запирающий сценарий и разблокирующий сценарий.

Запирающий скрипт — это обременение на выходе, которое определяет условия, выполнение которых разрешает трату выходов в будущем. Исторически сложилось, что запирающий сценарий был назван *scriptPubKey*, потому что обычно он хранил публичный ключ или Биткоин-адрес. В этой книге мы ссылаемся на него как на "запирающий скрипта" подразумевая гораздо более широкий спектр возможностей этих сценариев. В большинстве приложений то, что мы называем запирающим скриптом, в исходном коде называется *scriptPubKey*.

Разблокирующий сценарий — это такой, который "предлагает решение", или удовлетворяет условиям размещенным на выходе с запирающим скриптом и позволяет выходам быть потраченным. Разблокирующие скрипты являются частью каждого входа транзакции, и как правило они содержат цифровую подпись приватного ключа, оставленную кошельком пользователя. Исторически сложилось так, что сценарий разблокировки называют *scriptSig*, потому что обычно там содержится цифровая подпись. В коде большинства Биткоин-приложений сценарий разблокировки называется *scriptSig*. В этой книге мы ссылаемся на него как на "сценарий разблокировки", тем самым признавая более широкий спектр требований запирающих сценариев, потому что не все сценарии разблокировки должны содержать подписи.

Каждый Биткоин-клиент будет проверять транзакции, выполняя запирающий и отпирающий скрипты вместе. Для каждого входа транзакции, проверяющее ПО сначала извлечет UTXO, на которое ссылается вход. Этот UTXO содержит запирающий сценарий, определяющий условия, удовлетворение которых необходимо для траты. Далее проверяющее ПО возьмет отпирающий

сценарий, содержащийся во входе, который пытается потратить этот UTXO и выполнит оба сценария.

В оригинальном клиенте Bitcoin, отпирающие и запирающие скрипты конкатенировались и выполнялись последовательно. По соображениям безопасности, это было изменено в 2010 году, из-за уязвимости, которая позволяла неправильно сформированному отпирающему сценарию помещать данные в стек и повреждать запирающий скрипт. В текущей реализации, скрипты выполняются отдельно, а стек передается между двумя запусками, как описано далее.

Сначала выполняется разблокирующий сценарий. Если он выполнен без ошибок (например, не осталось "висящих" операторов), основной (не альтернативный) стек копируется и запускается запирающий скрипт. Если результат выполнения запирающего скрипта с данными стека, скопированными из отпирающего сценария дает значение "ИСТИНА", то отпирающий сценарий смог удовлетворить условиям, налагаемым запирающим сценарием и, следовательно, вход авторизован для траты UTXO. Если после выполнения комбинированного сценария остается какой-либо результат, кроме "ИСТИНА", то вход ошибочен, так как он не смог удовлетворить условия траты размещенные в UTXO. Обратите внимание, что UTXO перманентно записан в blockchain, и, следовательно, неизменен и не зависит от неудачных попыток потратить его по ссылке в новой транзакции. Только действительная транзакция, удовлетворяющая условиям UTXO приводит к пометке UTXO как "потраченного" и удаляется из набора доступных (неизрасходованных) UTXO.

В Комбинирование scriptSig и scriptPubKey для получения сценария транзакции пример отпирающих и запирающих сценариев для наиболее распространенного типа транзакции Биткоин (платежа на хэш публичного ключа), показывающий комбинированный сценарий результате конкатенации отпирающих и запирающих сценариев до проверки сценария.

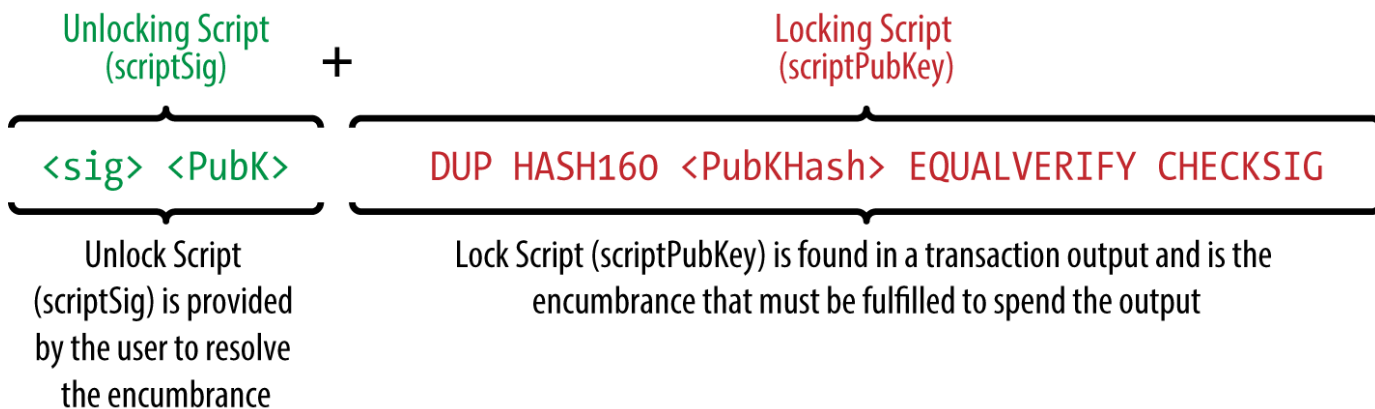


Figure 1. Комбинирование scriptSig и scriptPubKey для получения сценария транзакции

Язык сценариев

Язык сценариев Биткоин-транзакций называемый Script, является Forth-подобным языком с обратной польской нотацией и выполнением на основе стека. Если это звучит как бред, вы, вероятно, не изучали языки программирования 1960-х. Script — это очень простой язык, который был разработан ограниченным и выполняемым на оборудовании с возможностями карманного калькулятора. Это требует минимальной обработки и не позволяет делать многие

модные вещи, на которые способны современные языки программирования. В случае программируемых денег, это задумано преднамеренно для безопасности.

Скриптовый язык Биткоин, называется стековым, потому что он использует структуру данных под названием *стек*. Стек — это очень простая структура данных, которая может быть визуализирована в виде стопки карт. Стек позволяет две операции: добавление и удаление. Добавление оставляет элемент на вершине стека. Удаление извлекает верхний элемент из стека.

Язык сценариев выполняет скрипт по одному элементу слева направо. Числа (данные) помещаются на стек. Операторы помещают или извлекают один или несколько параметров из стека, производят действия над ними, и могут помещать результат выполнения обратно на стек. Например `OP_ADD` извлечет два элемента из стека, сложит их и поместит полученную сумму на стек.

Условные операторы вычисляют условия, производя булевский результат ИСТИНА или ЛОЖЬ. Например, `OP_EQUAL` извлекает два элемента из стека и помещает ИСТИНА (представлена числом 1), если они равны или ЛОЖЬ (представлена нулем), если они не равны. Сценарии Биткоин-транзакций обычно содержат условный оператор, так что они могут произвести истинный результат, что означает действительную транзакцию.

В [Проверка Биткоин-скрипта с простыми математическими операторами](#) сценарий `2 3 OP_ADD 5 OP_EQUAL` демонстрирует оператор арифметического сложения `OP_ADD`, складывающий два числа и помещающий результат на стек, где затем условный оператор `OP_EQUAL` проверяет, что результирующая сумма равна 5. Для краткости префикс `OP_` пропущен в пошаговом примере.

Ниже приводится несколько более сложный сценарий, который вычисляет $2 + 7 - 3 + 1$. Обратите внимание, что если скрипт содержит несколько операторов подряд, стек позволяет применить результат одного оператора к следующему оператору:

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Попробуйте проверить предыдущий сценарий самостоятельно, используя карандаш и бумагу. Когда скрипт выполнится, у вас на стеке должно остаться значение ИСТИНА.

Хотя в большинстве случаев запирающие скрипты ссылаются на Биткоин-адреса или публичный ключ, таким образом, требуя доказательства владения для траты средств, сценарий не обязательно должен быть таким сложным. Любая комбинация из запирающего и разблокирующего скриптов, которая дает в итоге значение ИСТИНА является действительным. Простая арифметика, которую мы использовали в качестве примера языка сценариев также может использоваться в качестве действительного запирающего сценария, который может быть использован для блокировки выхода транзакции.

Использование части примера сценария с арифметическими операциями в качестве

запирающего скрипта:

```
3 OP_ADD 5 OP_EQUAL
```

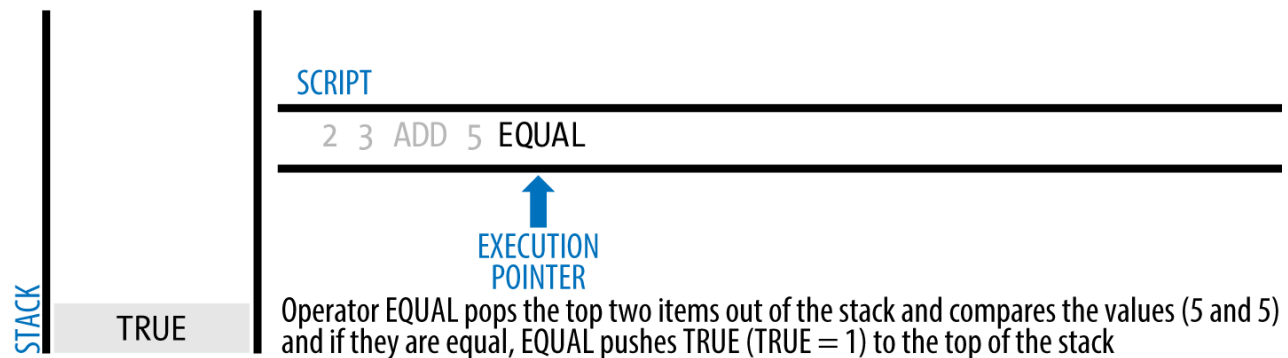
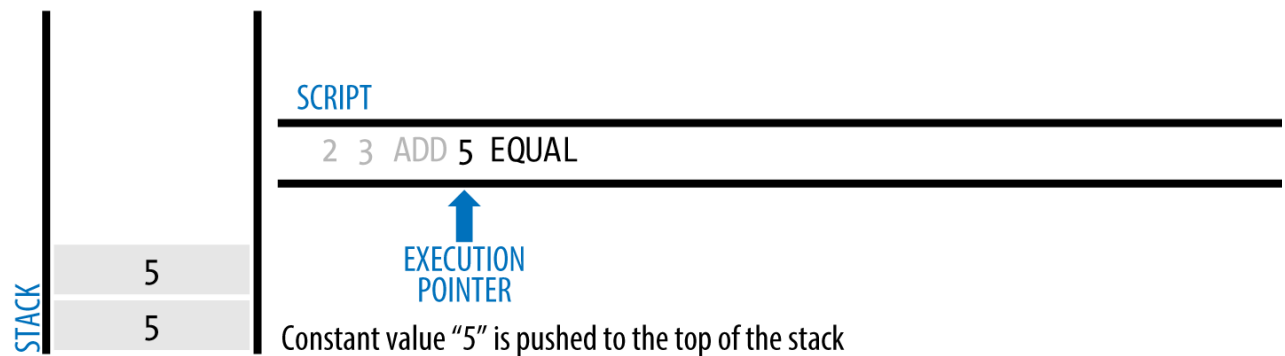
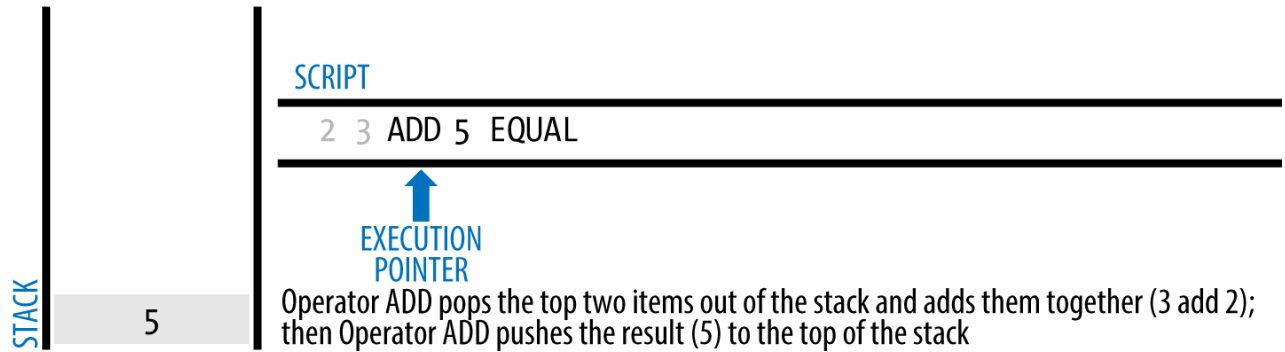
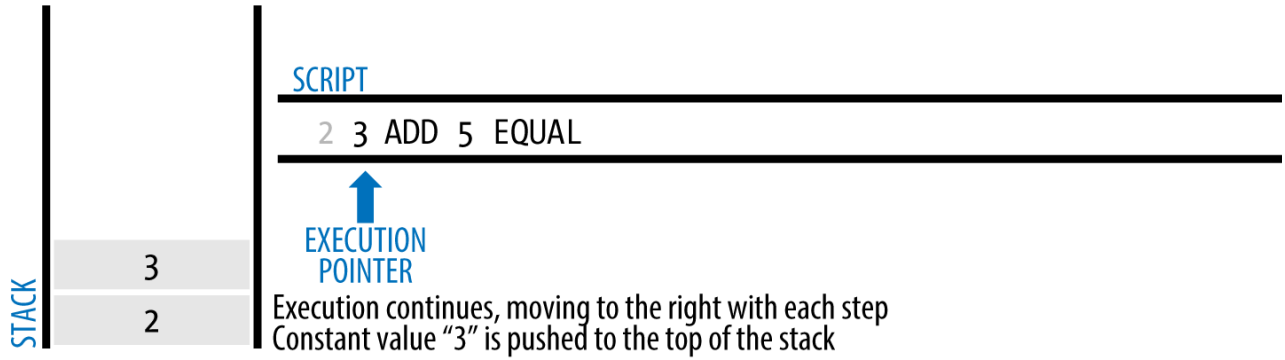
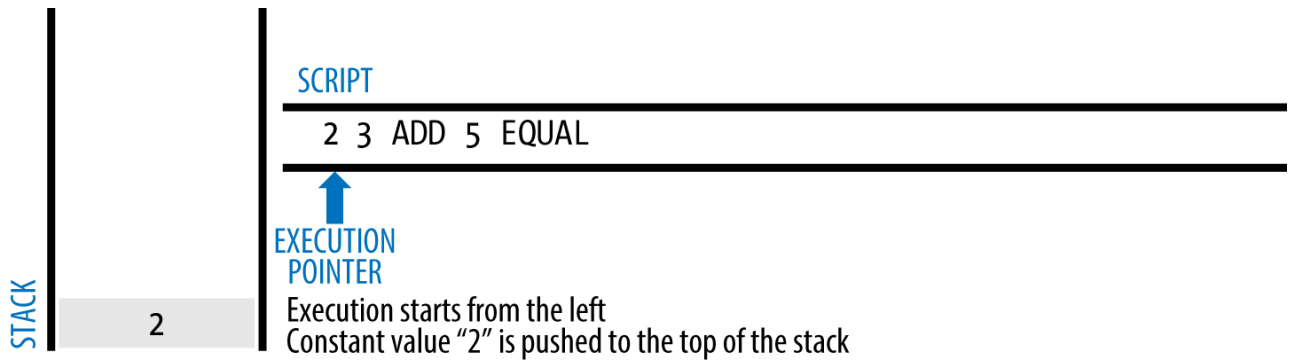
Этот сценарий может быть удовлетворен транзакцией, содержащей вход с таким сценарием разблокировки:

```
2
```

Проверяющее программное обеспечение объединяет запирающий и разблокирующий сценарии и в результате получается сценарий:

```
2 3 OP_ADD 5 OP_EQUAL
```

Как мы видели в пошаговом примере в [Проверка Биткоин-скрипта с простыми математическими операторами](#) когда этот скрипт выполняется, результат равен OP_TRUE, что делает транзакцию действительной. В результате UTXO могут быть потрачены кем угодно, кто обладает достаточными знаниями арифметики, чтобы понять, что число 2 удовлетворяет сценарию.



TIP

Транзакции действительны, если верхний результат на стеке имеет значение ИСТИНА (отмечены как `{0x01}`), любое другое ненулевое значение или если стек оказывается пуст после выполнения скрипта. Транзакции признаются недействительными, если верхним значение на стеке оказывается ЛОЖЬ (пустое значение нулевой длины, обозначаемое, как `{}`), или если выполнение скрипта явно останавливается оператором таким, как, например, `OP_VERIFY`, `OP_RETURN` или условным терминатором вроде `OP_ENDIF`. См. подробнее в [\[tx_script_ops\]](#).

Неполнота по Тьюрингу

Язык Биткоин-транзакций содержит много операторов, но сознательно ограничивается в одном важном вопросе — в нем нет циклов или сложных возможностей управления потоком выполнения. Это гарантирует, что язык не *Полный по Тьюрингу*, что означает ограничение сложности скриптов и предсказуемое время выполнения. Script не является языком общего назначения. Его ограничения гарантируют, что язык не может быть использован для создания бесконечного цикла или другой формы "логической бомбы", которая могла бы оказаться в транзакции, вызвав, таким образом атаку на отказ в обслуживании Биткоин-сети. Помните, каждая транзакция подтверждается каждым полным узлом в сети Биткоин. Ограничение языка предотвращает использование механизма проверки транзакций в качестве уязвимости.

Проверка без состояния

Язык сценариев Биткоин-транзакций не требует восстановления состояния до выполнения скрипта и состояние не сохраняется после выполнения. Таким образом, вся информация, необходимая для выполнения сценария содержится в самом сценарии. Сценарий будет предсказуемо выполнен одинаково в любой системе. Если ваша система нашла сценарий действительным, вы можете быть уверены, что любая другая системы в сети Биткоин также проверит сценарий и найдет его действительным, что означает, что действительная транзакция действительна для всех и все знают это. Эта предсказуемость результатов является важным преимуществом системы Биткоин.

Стандартные транзакции

В первые несколько лет развития Bitcoin, разработчики внесли некоторые ограничения на типы сценариев, которые могут быть обработаны эталонным клиентом. Эти ограничения закодированы внутри функции `isStandard()`, которая определяет пять типов "стандартных" операций. Эти ограничения носят временный характер и могут быть отменены к тому времени, когда вы прочитаете это. До тех пор, пять стандартных типов транзакционных сценариев являются единственными, которые будут приняты эталонным клиентом и большинством майнеров под управлением эталонного клиента. Несмотря на то, что существует возможность создать нестандартную транзакцию, содержащую сценарий, не

являющийся одним из стандартных типов, вам потребуется найти майнера, который бы ее пропустил и включил в блок.

В исходном коде клиента Bitcoin Core (базовая реализация), можно посмотреть какие сценарии разрешены в данный момент.

Пять стандартных типов сценариев транзакций: pay-to-public-key-hash (P2PKH), public-key, multi-signature (с ограничением на 15 ключей), pay-to-script-hash (P2SH), и выход данных (OP_RETURN). Опишем их подробнее в следующих разделах.

Pay-to-Public-Key-Hash (P2PKH)

Подавляющее большинство транзакций, обрабатываемых сетью Bitcoin — это P2PKH-транзакции. Они содержат запирающий сценарий, который обременяет выход хешем публичного ключа, более известный как адрес Bitcoin. Транзакции, которые платят под Биткойн-адресам содержат скрипты P2PKH. Выход запертый с помощью сценария P2PKH может быть отперт путем предоставления открытого ключа и цифровой подписи, полученной от соответствующего приватного ключа.

Например, давайте снова посмотрим на платеж Алисы в кафе Боба. Алиса заплатила 0,015 биткоинов на адрес кафе. Выход этой транзакции будет содержать блокирующий сценарий в виде:

```
OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUAL OP_CHECKSIG
```

Cafe Public Key Hash эквивалентен Bitcoin-адресу кафе, без кодирования Base58Check. Большинство приложений отображают *хеш публичного ключа* в шестнадцатеричном виде, а не в обычном для Bitcoin-адреса Base58Check формате, который начинается с "1".

Предыдущий запирающий сценарий может быть удовлетворен с помощью подобного отпирающего сценария:

```
<Cafe Signature> <Cafe Public Key>
```

Оба сценария вместе образуют следующий комбинированный сценарий проверки:

```
<Cafe Signature> <Cafe Public Key> OP_DUP OP_HASH160  
<Cafe Public Key Hash> OP_EQUAL OP_CHECKSIG
```

При выполнении этот комбинированный сценарий имеет результат TRUE тогда и только тогда, когда отпирающий сценарий удовлетворит условиям, установленным запирающим сценарием. Другими словами, результат будет TRUE, если отпирающий скрипт имеет правильную подпись приватным ключом кафе, что соответствует хешу публичного ключа,

установленного в качестве обременения.

На рисунках [<xref linkend="P2PubKHash1" xrefstyle="select: labelnumber"/>](#) и [<xref linkend="P2PubKHash2" xrefstyle="select: labelnumber"/>](#) показано (в двух частях) пошаговое выполнение комбинированного сценария, который проверяет правильность транзакции.

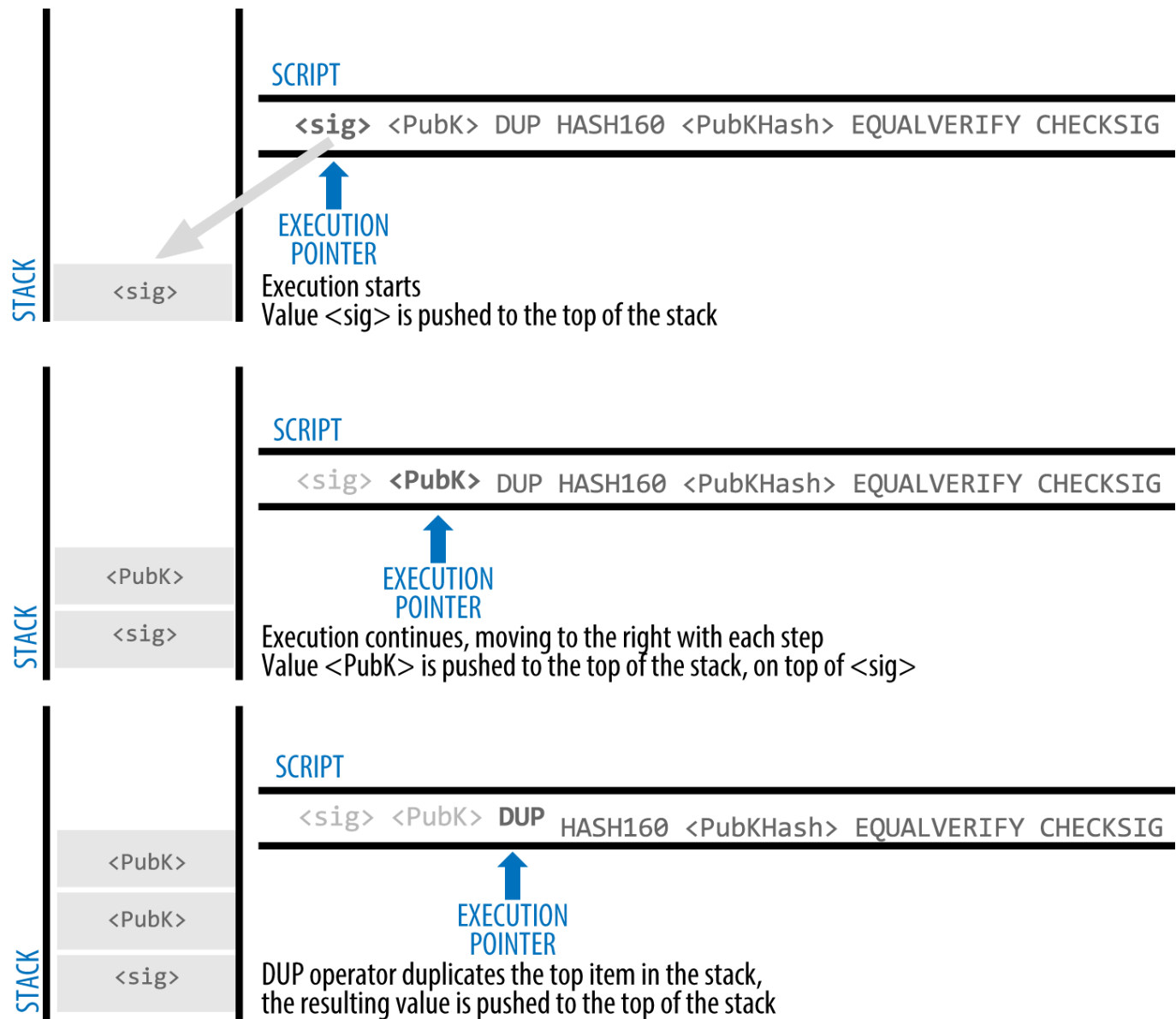


Figure 3. Выполнение сценария для P2PKH-транзакции (Часть 1 из 2)

Pay-to-Public-Key

Pay-to-public-key — это более простая форма платежа, чем pay-to-public-key-hash. В этом виде сценариев, в теле запирающего сценария хранится публичный ключ, а не более короткий хеш публичного ключа, как в случае с P2PKH (Pay-to-public-key-hash), рассмотренном ранее. Pay-to-public-key-hash был введен Сатоши в целях сделать Bitcoin-адреса короче, для упрощения использования. Pay-to-public-key в настоящее время наиболее часто встречается в coinbase-транзакциях, создаваемых старым майнинговым ПО.

Запирающий сценарий pay-to-public-key выглядит следующим образом:

```
<Public Key A> OP_CHECKSIG
```

Соответствующий отпирающий скрипт для этого типа выхода — это простая подпись:

```
<Signature from Private Key A>
```

Комбинированный сценарий, который проверяется программным обеспечением проверки транзакций:

```
<Signature from Private Key A> <Public Key A> OP_CHECKSIG
```

Этот сценарий представляет собой простой вызов оператора CHECKSIG, который проверяет соответствие подписи ключу и возвращает TRUE на стеке.

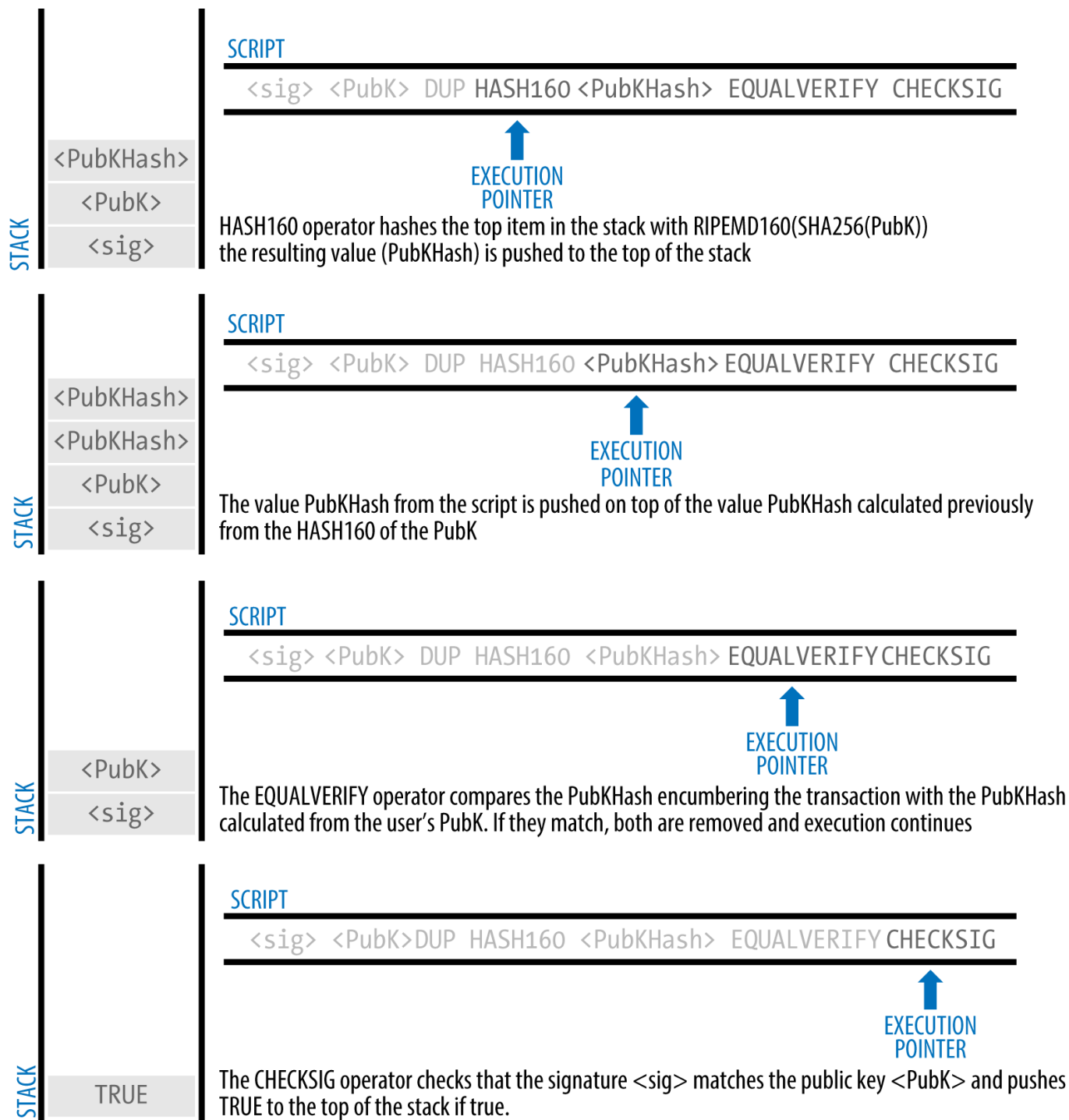


Figure 4. Выполнение сценария для P2PKH-транзакции (Часть 2 из 2)

Множественная Подпись

Мульти-подписные скрипты устанавливают условие, что из N публичных ключей, перечисленных в сценарии, по меньшей мере M должны предоставить подписи для освобождения обременения. Эта схема также известна как M -из- N , где N представляет собой общее количество ключей, а M представляет собой порог количества подписей, необходимых для валидации. Например, в случае мульти-подписи 2-из-3, три публичных ключа перечисляются в качестве потенциальных подписантов и по крайней мере два из них должны

быть использованы для создания действительной подписи транзакции, освобождающей эти средства. "multi-signature scripts","limits on")В настоящий момент, стандартные мульти-подписные сценарии ограничены максимумом в 15 публичных ключей, т.е. вы можете использовать мульти-подписи в диапазоне от 1-из-1 до 15-о-15. Ограничение в 15 ключей может быть увеличено к моменту, когда эта книга увидит свет, так что за подробностями обратитесь к документации на функцию .

Общий вид запирающего сценария с условием мульти-подписи М-из-N:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N OP_CHECKMULTISIG
```

где N — общее число перечисленных публичных ключей, M — порог количества необходимых подписей.

Запирающий сценарий условия мульти-подписи 2-из-3 выглядит следующим образом:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 OP_CHECKMULTISIG
```

Предыдущий запирающий сценарий может быть удовлетворен отпирающим сценарием, содержащим пары подписей и публичных ключей:

```
OP_0 <Signature B> <Signature C>
```

или любой комбинацией из двух подписей секретных ключей, соответствующих трем перечисленным публичным ключам.

NOTE

Префикс OP_0 требуется для обхода ошибки в оригинальной реализации CHECKMULTISIG и игнорируется.

Два сценария вместе дают объединенный сценарий проверки:

```
OP_0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3  
OP_CHECKMULTISIG
```

При выполнении этот комбинированный сценарий даст в результате TRUE тогда и только тогда, когда отпирающий скрипт будет удовлетворять условиям, установленным запирающим сценарием. В таком случае условие состоит в требовании предоставления валидной подписи двух приватных ключей, которые соответствуют двум из трех публичных ключей.

Вывод данных (OP_RETURN)

Распределенная бухгалтерская книга Bitcoin потенциально обладает возможностями

использования далеко за пределами только лишь платежной системы. Многие разработчики пытались использовать язык сценариев транзакций, чтобы воспользоваться преимуществами безопасности и устойчивости системы для приложений, таких как цифровые нотариальные услуги, акционерные сертификаты и умные контракты. Ранние попытки использовать язык сценариев в подобных целях подразумевал создание выходов транзакций, записывающих данные в blockchain; например, для записи цифрового отпечатка файла таким образом, что любой мог бы получить доказательство подтверждающие факт существования этого файла на определенную дату путем ссылки на эту транзакцию.

Использование blockchain биткоина для хранения данных не связанных с биткоин-платежами является спорным вопросом. Многие разработчики считают такое использование оскорбительным и стараются препятствовать ему. Другие рассматривают его как демонстрацию мощных возможностей технологии blockchain и хотели бы поощрять такое экспериментирование. Те, кто возражает против включения данных не относящихся к платежам, утверждают, что это приводит к "раздуванию blockchain" отягощая полные узлы, увеличивая стоимость дискового хранения данных, которые не предназначены для хранения в blockchain. Кроме того, такие операции создают UTXO, которые не могут быть потрачены, используя 20-ти байтовое поле адреса назначения в свободной форме. Поскольку адрес используется для данных, не существует соответствующего приватного ключа и полученный UTXO *никогда* не может быть потрачен; это поддельный платеж. Подобные транзакции, которые никогда не могут быть потрачены, потому никогда не удаляются из набора UTXO и приводят к бесконечному "раздуванию" размера базы данных UTXO.

В версии 0.9 клиента Bitcoin Core, был достигнут компромисс путем введения оператора OP_RETURN. OP_RETURN позволяет разработчикам добавлять 80 байт данных к выходу транзакции. Тем не менее, в отличие от использования "поддельных" UTXO, оператор OP_RETURN явно создает *доказуемо неспособный быть потраченным* выход, который не нужно хранить в наборе UTXO. Выходы OP_RETURN записываются в blockchain, поэтому они потребляют дисковое пространство и способствуют увеличению размера blockchain, но они не хранятся в наборе UTXO и, следовательно, не раздувают пул памяти UTXO и не требуют от полных узлов больше RAM.

Скрипты OP_RETURN выглядят следующим образом:

```
OP_RETURN <data>
```

Часть данных ограничена 80-ю байтами, и наиболее часто представляет собой хэш, такой как выход алгоритма SHA256 (32 байта). Многие приложения ставят префикс перед данными для идентификации приложения. Например, [Proof of Existence](#) служба цифровых нотариусов использует 8-байтовый префикс "DOCPROOF", который из ASCII кодируется в шестнадцатеричный формат в виде 44f4350524f4f46.

Имейте в виду, что не существует "отпирывающего сценария", который бы соответствовал OP_RETURN и который мог бы быть использован для "траты" выхода OP_RETURN. Весь смысл OP_RETURN состоит в том, что вы не можете потратить средства запертые в этом выходе, и,

следовательно, его не нужно держать в наборе UTXO, в качестве потенциально расходуемого — `OP_RETURN` *доказуемо нерасходуемый*. `OP_RETURN` — это как правило выход с нулевым количеством биткоинов, так как любые биткоины, назначенные на такой выход фактически теряются навсегда. Если `OP_RETURN` встречается процедурой валидации сценариев, то это сразу же приводит к остановке выполнения сценария и пометке транзакции недействительной. Таким образом, если вы случайно сошлетесь на `OP_RETURN` выход в качестве входа транзакции, это сделает транзакцию недействительной.

Стандартная транзакция (такая, которая проходит проверку `isStandard()`) может иметь только один выход `OP_RETURN`. Однако, один выход `OP_RETURN` может быть объединен в транзакции с выходами любого другого типа.

Начиная с версии 0.10 к опциям командной строки Bitcoin Core были добавлены два новых параметра. Опция `datacarrier` контролирует ретрансляцию и майнинг `OP_RETURN` транзакций и по умолчанию разрешена (установлена в "1"). Опция `datacarriersize` принимает числовой аргумент, указывающий максимальный размер в байтах данных `OP_RETURN`, по умолчанию это число составляет 40 байт.

NOTE

Изначально предполагалось, что данные `OP_RETURN` будут иметь максимальный предел в 80 байтов, но на момент запуска этой возможности это значение было уменьшено до 40 байт. В феврале 2015-го года, в Bitcoin Core версии 0.10, этот предел был поднят обратно в 80 байт. Узлы вполне не ретранслировать или майнить `OP_RETURN`, либо ретранслировать и майнить только `OP_RETURN` содержащие данные размером менее 80 байт.

Pay-to-Script-Hash (P2SH)

Pay-to-script-hash (P2SH) были введены в 2012 году в качестве нового мощного типа транзакции, что значительно упростило использование сложных сценариев транзакций. Для того, чтобы объяснить необходимость P2SH, давайте рассмотрим практический пример.

В [\[ch01_intro_what_is_bitcoin\]](#) мы познакомились с Мохаммедом, импортером электронной аппаратуры из Дубая. Компания Мохаммеда широко использует функцию мульти-подписи Bitcoin. Сценарии мульти-подписи — это очень мощный инструмент и один из наиболее распространенных видов использования передовых возможностей сценариев Bitcoin. Компания Мохаммеда использует сценарии мульти-подписи для всех платежей от клиентов, что в терминах бухгалтерии называется "дебиторской задолженностью". С помощью схемы с несколькими подписями, любые платежи, сделанные клиентами заблокированы таким образом, что требуется по крайней мере, две подписи, Мохаммеда (или его доверенного лица, обладающего резервным ключом) и одного из его партнеров. Схема с мульти-подписью предлагает средства корпоративного управления и защищает от кражи, хищения или потери средств.

Результирующий сценарий довольно длин и выглядит следующим образом:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> <Attorney Public Key> 5 OP_CHECKMULTISIG
```

Несмотря на то, мульти-подписные сценарии являются мощной возможностью, она громоздка в использовании. С учетом предыдущего сценария, Мухаммед должен был бы сообщить этот сценарий каждому клиенту до оплаты. Каждый клиент должен будет использовать специальный Bitcoin-кошелек с возможностью создания пользовательских сценариев транзакций, и каждый клиент должен был бы понимать, как создавать транзакцию с помощью кастомных сценариев. Кроме того, результирующая транзакция будет примерно в пять раз больше простой из-за большой длины открытых ключей. Бремя этой очень крупной сделки будет нести заказчик в виде комиссионных. И, наконец, большой сценарий, подобный этому, должен храниться в UTXO, в оперативной памяти каждого полного узла до тех пор, пока он не будет потрачен. Все эти проблемы делают использование сложных выходных скриптов трудным на практике.

Pay-to-script-hash (P2SH) был разработан для решения этих практических трудностей и для того, чтобы сделать использование сложных скриптов таким же простым, как платеж на Bitcoin-адрес. В платежах P2SH, сложный запирающий сценарий заменяется его цифровым отпечатком, криптографическим хэшем. Когда позднее появляется тратящая транзакция, она должна содержать сценарий с соответствующим хэшем, в дополнение к отпирающему сценарию. Проще говоря, P2SH означает "заплатить по сценарию, соответствующему этому хэшу, который будет представлен позже, в момент, когда этот выход будет потрачен."

В P2SH-транзакциях, запирающий сценарий заменяется хэшем и называется *погашающим сценарием* потому, что он представлен системе в момент погашения, а не в качестве запирающего сценария. В [Сложный скрипт без P2SH](#) показан сценарий без P2SH и в [Комплексный сценарий в качестве P2SH](#) показан тот же сценарий, закодированный с P2SH.

Table 4. Сложный скрипт без P2SH

Запирающий скрипт	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG
Отпирающий скрипт	Sig1 Sig2

Table 5. Комплексный сценарий в качестве P2SH

Погашающий скрипт	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG
Запирающий скрипт	OP_HASH160 <20-байтный хэш погашающего скрипта> OP_EQUAL
Отпирающий скрипт	Sig1 Sig2 погашающий скрипт

Как вы можете видеть из таблиц, с P2SH сложный сценарий, который подробно описывает условия траты выхода (сценарий погашения) не представлен в запирающем сценарии. Вместо

этого, в запирающем сценарии представлен его хеш, а погашающий сценарий представляется позднее в отпирающем сценарии, когда расходуется выход. Это перекладывает бремя комиссионных и сложности с отправителя на получателя транзакции.

Давайте посмотрим на компанию Мухаммеда, комплексный сценарий мульти-подписи, и результирующие P2SH скрипты.

Во-первых, сценарий мульти-подписи, который компания Мухаммеда использует для всех входящих платежей от клиентов:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3 Public Key> <Attorney Public Key> 5 OP_CHECKMULTISIG
```

Если пробелы заменить публичными ключами (которые показаны здесь в виде 520-разрядных чисел, начинающихся с 04), то можно заметить, что сценарий становится очень длинным:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC6984
D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308
EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49047E632
48B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9162F0279CFC1
0F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9
D85BAAA93A4AB3A8F044DADA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1E
CED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D1
37AAB59E0B000EB7ED238F4D800 5 OP_CHECKMULTISIG
```

Вместо этого, весь этот сценарий может быть представлен 20-байтовым криптографическим хэшем, полученным применением алгоритма хэширования SHA256, с последующим применением алгоритма RIPEMD160. Вот так будет выглядеть 20-байтовый хэш из предыдущего сценария:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

P2SH транзакция запирает выход на этот хэш вместо более длинного сценария, с помощью запирающего сценария:

```
OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL
```

что, как можно видеть, намного короче. Вместо того, чтобы "заплатить на этот 5-ти ключевой сценарий мульти-подписи," эквивалентная P2SH-транзакция звучит как "заплатить на скрипт с этим хэшем". Клиент, делающий оплату компании Мохаммеда в своем платеже должен включать в себя только этот более короткий запирающий сценарий. Когда Мухаммед хочет

потратить этот UTXO, он должен предоставить оригинальный выкупающий сценарий (тот, хэшем которого был заперт UTXO) и подписи, необходимые для его разблокирования, вроде этого:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG>
```

Оба сценария объединяются в два этапа. Во-первых, погашающий сценарий проверяется отпиранием сценарием на предмет совпадения хеша:

```
<2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG> OP_HASH160 <погашающий scriptHash> OP_EQUAL
```

Если хеш погашающего сценария совпадает, отпирющий скрипт выполняется самостоятельно и разблокирует погашающий сценарий:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG
```

Pay-to-script-hash адреса

Еще одной важной частью P2SH является возможность кодировать хэш сценария в качестве адреса, как это определено в BIP0013. P2SH адреса — это 20-ти байтовый хэш сценария в кодировке Base58Check, точно так же, как Bitcoin-адрес — это 20-байтовый хэш открытого ключа в кодировке Base58Check. P2SH-адреса используют префикс версии "5", что приводит к Base58Check-кодированным адресам, которые начинаются с "3". Например, сложный сценарий Мухаммеда, хешированный и в кодировке Base58Check в виде P2SH-адреса становится 39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw. Теперь, Мохаммед может дать этот "адрес" своим клиентам, и они могут использовать практически любой Bitcoin кошелек, чтобы сделать простой платеж, как если бы это был Bitcoin адрес. Префикс 3 дает им подсказку, что это особый тип адреса, соответствующий скрипту вместо открытого ключа, но работающий точно так же, как платеж на адрес Bitcoin.

P2SH адреса скрывают все сложности, так что человек, совершающий платеж не видит сценарий.

Преимущества pay-to-script-hash

Функция pay-to-script-hash предлагает следующие преимущества по сравнению с непосредственным использованием сложных сценариев для записывания выходов:

- Сложные сценарии заменяются на более хеши на выходе транзакции, что делает размер транзакций меньше.
- Сценарии могут быть закодированы в виде адреса, так что отправителю и кошельку отправителя не требуется сложной реализации P2SH.

- P2SH перекладывает бремя построения сценария на получателя, а не отправителя.
- P2SH перекладывает бремя хранения данных для длинного сценария с выхода (который находится во множестве UTXO) на вход (который хранится в blockchain).
- P2SH перекладывает бремя хранения данных для длинного сценария с настоящего времени (момента платежа) на будущее (момент распоряжения средствами).
- P2SH переносит оплату комиссионных за длинный сценарий с отправителя на получателя, который должен будет включить длинный погашающий сценарий.

Погашающий сценарий и валидация isStandard

До версии 0.9.2 клиента Bitcoin Core, pay-to-script-hash был ограничен стандартными типами сценариев транзакций с помощью функции isStandard(). Это означает, что погашающий сценарий представленный в тратящей транзакции может быть только одного из стандартных типов: P2PK, P2PKH, или мульти-подписной, за исключением OP_RETURN и самого P2SH.

Начиная с версии 0.9.2 клиента Bitcoin Core, P2SH-транзакции могут содержать любой валидный сценарий, что делает стандарт P2SH гораздо более гибким и позволяет эксперименты со многими новыми и сложными типами транзакций.

Обратите внимание, что поместить P2SH-сценарий внутрь погашающего P2SH-сценария не получится, поскольку спецификация P2SH не позволяет рекурсию. Вы также не сможете использовать OP_RETURN в погашающем сценарии, так как OP_RETURN не может быть погашен по определению.

Обратите внимание, что, так как погашающий сценарий не представлен в сети до тех пор, пока вы не попытаетесь потратить выход P2SH, если заблокировать выход хэшем недействительной транзакции, он все равно будет обработан. Тем не менее, вы не сможете его потратить, так как расходная транзакция, которая включает в себя погашающий сценарий, не будет принята, так как этот сценарий недействителен. Это создает риск, потому что вы можете заблокировать биткоины в P2SH, который позднее нельзя будет израсходовать. Сеть примет обременение P2SH, даже если оно соответствует недопустимому погашающему сценарию, так как хэш сценария не дает никаких указаний по поводу сценария, который он представляет.

WARNING

P2SH-запирающие скрипты содержат хэш выкупающего сценария, который не дает никаких намеков относительно содержания самого скрипта. P2SH-транзакция будет считаться валидной и будет принята, даже если выкупающий сценарий является недействительным. Подобным образом вы можете так случайно заблокировать биткоины, что они позднее не смогут быть потрачены.

Сеть Биткойн

Децентрализованная Сетевая Архитектура

Bitcoin структурирован в виде одноранговой сети поверх сети Интернет. Термин одноранговая сеть, или P2P, означает, что компьютеры этой сети соединены непосредственно друг с другом, между собой равны и не существует никаких "особых" узлов, а также, что все узлы обеспечивают одинаковую функциональность. В подобной "плоской" топологии нет сервера, нет централизованной службы, и никакой иерархии в пределах сети: узлы взаимно предоставляют и потребляют услуги. Одноранговые сети от природы устойчивы, децентрализованы и открыты. Выдающимся примером P2P-архитектуры был сам ранний Интернет, где узлы IP-сети были равны между собой. Современная архитектура Интернет более иерархическая, но IP-протокол в основе все еще сохраняет свою плоскую топологию. Помимо Bitcoin, самое большой и успешное применение P2P технологий — это файл-шаринг, эра которого началась с Napster и продолжается в лице BitTorrent.

P2P-архитектура в основе Bitcoin это не просто топология. Bitcoin — это пиринговая система цифровых денег и сетевая архитектура является одновременно отражением и основой для этой базовой характеристике. Децентрализация управления находится в основании дизайна, а это может быть достигнуто и поддерживаться только плоской, P2P-сетью децентрализованного консенсуса.

Термин "сеть Bitcoin" относится к коллекции узлов, работающих под управлением протокола Bitcoin P2P. В дополнение к P2P-протоколу, существуют и другие протоколы, такие как Stratum, которые используются для майнинга и в легких или мобильных кошельках. Эти дополнительные протоколы обеспечиваются специальными шлюзами. Например, серверы Stratum обеспечивают двустороннюю связь майнинговых Stratum-узлов по протоколу Stratum с основной сетью Bitcoin. Для описания этой, более широкой сети, включающей в себя P2P-протокол, протоколы майнинговых пулов, протокол Stratum и любые другие соответствующие протоколы, соединяющие компоненты системы Bitcoin, мы используем термин "расширенная сеть Bitcoin".

Типы и роли узлов

Хотя узлы в P2P-сети Биткойн равнозначны, они могут взять на себя разные роли в зависимости от той функциональности, которую они поддерживают. Биткойн-узел представляет собой набор функций: маршрутизации, базы данных blockchain, майнинга и кошелька. Полный узел со всеми этими функциями показан на [Узел сети Биткойн, выполняющий все четыре функции: кошелек, майнер, база данных blockchain и сетевой маршрутизатор](#).

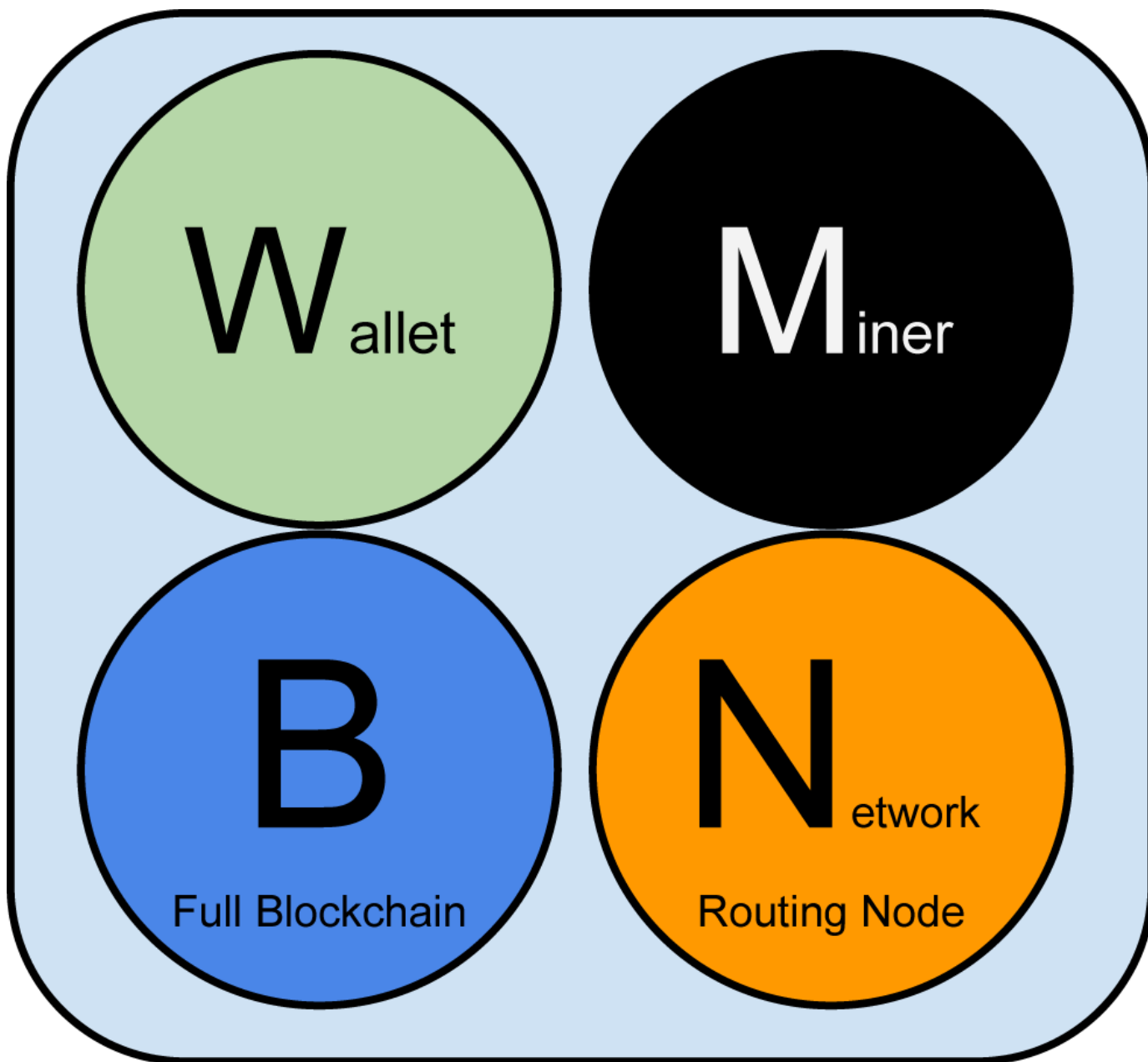


Figure 1. Узел сети Биткоин, выполняющий все четыре функции: кошелек, майнер, база данных blockchain и сетевой маршрутизатор

Все узлы поддерживают функцию маршрутизации для участия в сети и могут включать в себя другие функциональные возможности. Все узлы проверяют и распространяют транзакции и блоки, а также обнаруживают и поддерживают связь с соседями. В примере полного узла в [Узел сети Биткоин, выполняющий все четыре функции: кошелек, майнер, база данных blockchain и сетевой маршрутизатор](#), функция маршрутизации обозначается оранжевым кругом под названием "Узел Сетевой Маршрутизации".

Некоторые узлы, называемые полными узлами, также поддерживают полную и актуальную копию blockchain. Полные узлы могут автономно и авторитетно проверять любые транзакции. Некоторые узлы поддерживают только подмножество blockchain и проверяют транзакции методом, называемым *упрощенная проверка платежа* или SPV (simplified payment verification). Эти узлы называются SPV или легкими узлами. На схеме примера полного узла, функция

полной базы данных blockchain обозначается синим кругом под названием "Полный Blockchain." В [Расширенная сеть Bitcoin, показывающая различные типы узлов, шлюзов и протоколов](#), SPV-узлы нарисованы без синего круга, показывая, что они не содержат полную копию blockchain.

Майнинговые узлы конкурируют между собой в создании новых блоков. Некоторые из них также могут выступать полными узлами, поддерживая полную копию blockchain, в то время как другие представляют собой легкие узлы, участвующие в майнинговых пулах и зависящие от сервера пула, поддерживающего полный узел. Функция майнинга показана в полном узла в виде черного круга с названием "Майнер."

Пользовательские кошельки могут быть частью полного узла, как это обычно бывает с настольными Bitcoin-клиентами. Все чаще кошельки пользователей, особенно работающие на устройствах с ограниченными ресурсами, таких как смартфоны, являются SPV-узлами. Функция кошелька показана на [Узел сети Биткоин, выполняющий все четыре функции: кошелек, майнер, база данных blockchain и сетевой маршрутизатор](#) в виде зеленого круга с текстом "кошелек".

В дополнение к основным видам узлов в P2P-протоколе Биткоин, существуют сервера и узлы использующие другие протоколы, такие как специализированные протоколы майнинговых пулов и протоколы доступа для легких клиентов.

На рисунке [Различные типы узлов в расширенной сети Биткоин](#) показаны наиболее распространенные типы узлов расширенной сети Биткоин.

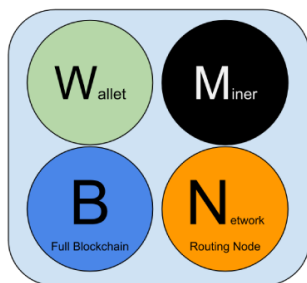
Расширенная сеть Bitcoin

Главная сеть, работающая по P2P-протоколу Bitcoin, состоит из от 7000 до 10000 узлов, работающих под управлением различных версий базового клиента (Bitcoin Core) и нескольких сотен узлов, работающих на основе различных других реализаций протокола Bitcoin таких как BitcoinJ, Libbitcoin, и btcd. Небольшой процент участников сети представляют собой майнинговые узлы, проверяющие транзакции и создающие новые блоки. Различные крупные компании работают с сетью Bitcoin, с помощью клиентов на основе Bitcoin Core, содержащими полные копии blockchain, но без функции майнинга или функции кошелька. Эти узлы работают в качестве маршрутизаторов, позволяя различным другим сервисам (биржам, кошелькам, проводникам, платежным шлюзам) существовать поверх.

Расширенная сеть Bitcoin включает в себя сеть, работающую по P2P-протоколу Bitcoin, описанную ранее, а также узлы, работающие при помощи специализированных протоколов. Параллельно основной сети существует некоторое количество пул-серверов и протокольных шлюзов, соединяющих узлы с другими протоколами: узлы майнинговых пулов (см. [\[ch8\]](#)) и легкие кошельки, которые не содержат в себе полную копию blockchain.

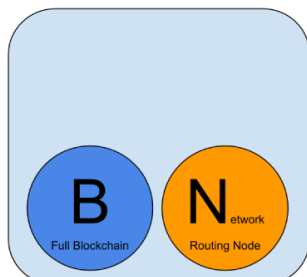
В [Расширенная сеть Bitcoin, показывающая различные типы узлов, шлюзов и протоколов](#) показана расширенная сеть Bitcoin с различными типами узлов, шлюзами, и клиентскими кошельками, а также различные протоколы, которые они используют для соединения друг с

другом.



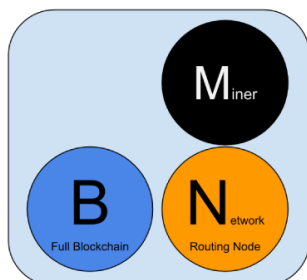
Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



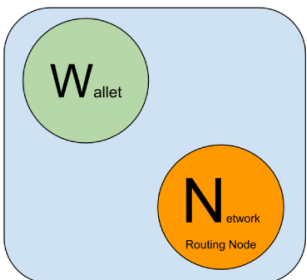
Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



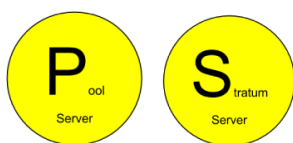
Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



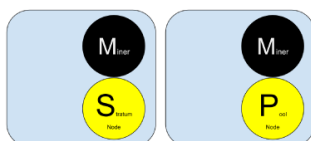
Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



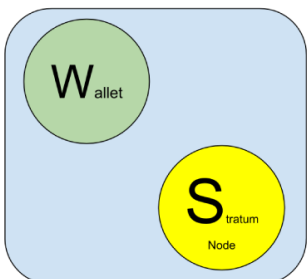
Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.



Mining Nodes

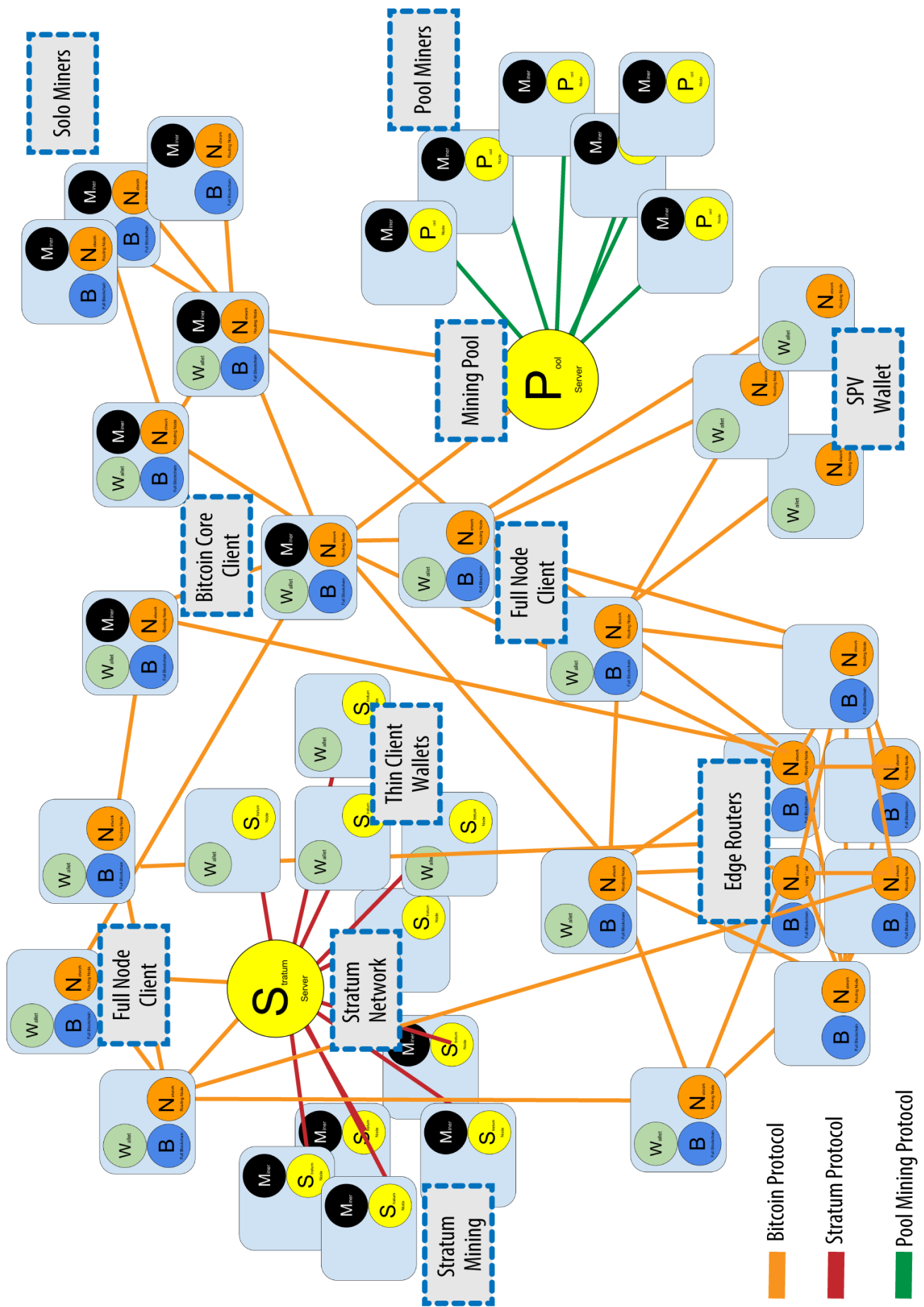
Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

Figure 2. Различные типы узлов в расширенной сети Биткоин



Обнаружение сети

Когда новый узел входит в строй, он должен обнаружить в сети другие узлы Bitcoin, по крайней мере один и, затем, подключиться к нему. Географическое расположение других узлов не имеет значения; топология Bitcoin-сети географически не определена. Таким образом, любые подключенные узлы могут быть выбраны случайным образом.

Для подключения к известному узлу, требуется установить TCP-соединение, как правило, к порту 8333 (порт по умолчанию), или альтернативному порту, если он предусмотрен. После установления соединения, узел начнет процесс установления сессии, т.н. "рукопожатие" (см. [Начальное рукопожатие между пирами](#)) путем передачи сообщения version, содержащего основную идентификационную информацию, в том числе:

PROTOCOL_VERSION

Константа, определяющая версию P2P-протокола, которую "понимает" Биткоин-клиент (например, 70002)

nLocalServices

Список локальных сервисов, поддерживаемых узлом, в настоящее время это только NODE_NETWORK

nTime

Текущее время

addrYou

IP-адрес удаленного узла так, как он виден с данного узла

addrMe

IP-адрес локального узла так, как его определил сам локальный узел

subver

Подверсия, содержащая тип и название программного обеспечения этого узла (например, "/Satoshi:0.9.2.1/")+

BestHeight

Высота блоков blockchain этого узла

(См. [GitHub](#) для примера версии сетевого сообщения version.)

Узел отвечает verack и опционально в ответ может послать сообщение version со своей собственной версией в случае, если предполагается использование этого же соединения в собственных целях.

Как новый узел находит остальные пиры? Первый метод состоит в запросе некоторого количества инициализирующих DNS-серверов, которые поддерживают список IP-адресов Bitcoin узлов. Некоторые из этих DNS-серверов работают под управлением модифицированной реализации BIND (Berkeley Internet Name Daemon) и по запросу возвращают случайное подмножество списка Bitcoin адресов узлов. Клиент Bitcoin Core содержит адреса пяти различных DNS-серверов и этим можно управлять при помощи опции `-dnsseed` (устанавливается в 1 (также по умолчанию) для использования инициализирующих DNS-серверов).

В качестве альтернативы, совершенно новый узел, который ничего не знает о сети, должен получить IP-адрес по крайней мере одного Bitcoin-узла, который сможет "познакомить" его с остальной сетью. Аргумент командной строки `-seednode` может быть использован для подключения к одному узлу только для "знакомства". После того, как "знакомства" состоялись, клиент отключится от этого узла и установит соединения с вновь обнаруженными узлами.

Node A

Node B

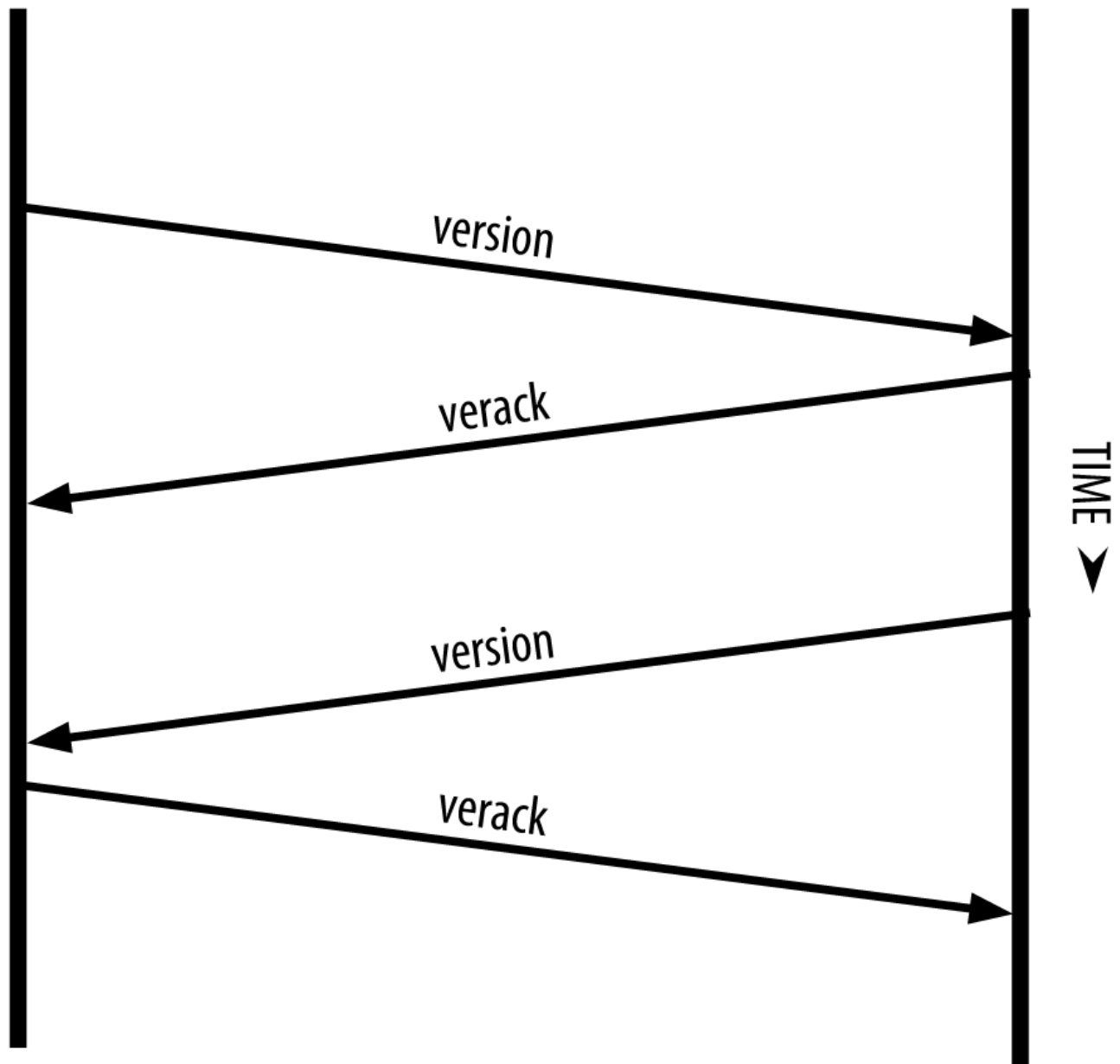


Figure 4. Начальное рукопожатие между пирами

После того, как одно или несколько соединений установлены, новый узел пошлет сообщение `addr`, содержащее собственный IP адрес своим соседям. Соседи, в свою очередь, перенаправят сообщение `addr` уже своим соседям, обеспечивая, вновь подключенному узлу лучшую известность. Кроме того, недавно подключенный узел может отправить `getaddr` соседям с просьбой вернуть список IP-адресов других узлов. Таким образом, узел может найти пиры для подключения и заявить о своем существовании в сети. В [Распространение адреса и обнаружение](#) показан протокол обнаружения адреса.

Node A

Node B

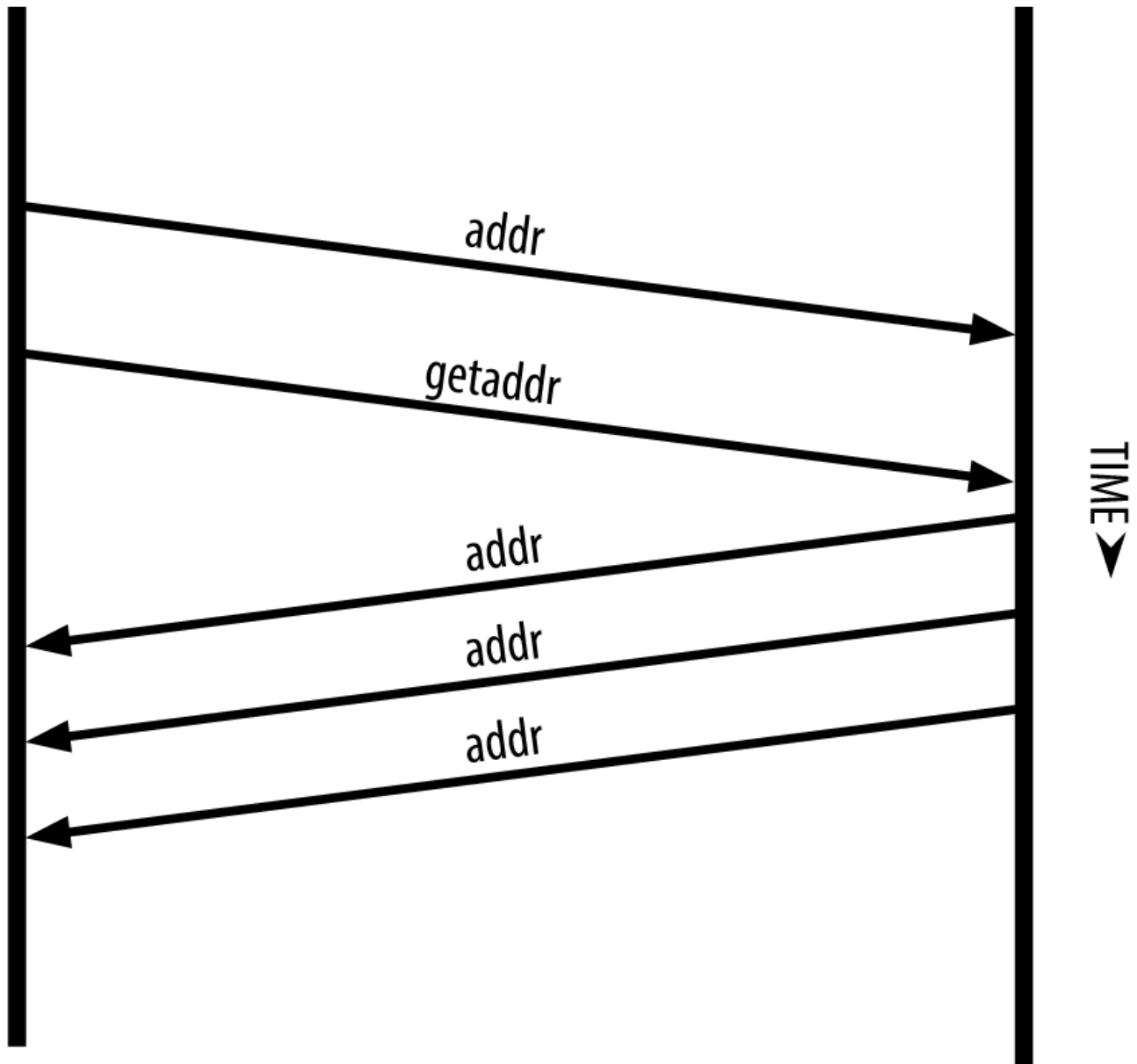


Figure 5. Распространение адреса и обнаружение

Узел должен соединяться с несколькими различными пирами для того, чтобы установить различные пути в сети Bitcoin. Эти пути не являются надежными — узлы приходят и уходят, и поэтому узел должен продолжать открывать новые узлы по мере утери старых связей, а также помогать другим узлам при первоначальной загрузке. Для первоначальной загрузки требуется только одно соединение, поскольку первый узел может представить узел своим пирам, а те в свою очередь уже своим. После первоначальной загрузки, узел будет помнить о своих самых последних успешных пирах, так что если он перезагрузится, то сможет быстро восстановить предыдущие связи. Если ни один из бывших пиров не отвечает, узел может использовать исходные узлы для новой попытки первоначальной загрузки.

На узле с запущенным клиентом Bitcoin Core, вы можете получить список соединений 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 отключить автоматическое управление пирами и указать список IP-адресов, может использоваться опция `-connect=<IPAddress>`, в которой можно перечислить один или несколько IP-адресов узлов. Если используется эта опция, узел будет подключен только к выбранным адресам, вместо того, чтобы автоматически открывать и поддерживать соединения с другими узлами.

Если через соединение не приходит трафика, узлы будут периодически обмениваться сообщениями для поддержки связи. Если от узла ничего не слышно в течение более 90 минут, то предполагается, что он отключился и можно начинать поиски замены. Таким образом, сеть динамически реагирует на изменения и может органически увеличиваться и уменьшаться по мере необходимости без какого-либо центрального контроля.

Полные Узлы

Полные узлы — это узлы, содержащие полную копию blockchain со всеми транзакциями. Если быть точным их, вероятно, следует называть "полными blockchain-узлами". В первые годы существования Bitcoin, все узлы были полными и в настоящее время клиент Bitcoin Core представляет собой полный blockchain узел. В течение последних двух лет, однако, получили хождение новые виды клиентов, не содержащих полный blockchain и работающих как легкие клиенты. Мы рассмотрим их более подробно в следующем разделе.

Полные узлы поддерживают полную и всегда свежую копию цепочки блоков Биткоин со всеми транзакциями, которые они независимо друг от друга собирают и верифицируют, начиная с самого первого блока (блока генезиса) вплоть до последнего известного сети блока. Полный узел может самостоятельно и авторитетно проверить любую транзакцию, не прибегая или полагаясь на другие узлы или источники информации. Полный узел получает сообщения о новых блоках транзакций из сети, которые он затем проверяет и включает в свою локальную копию blockchain.

Вы можете просто убедиться, что используете полный узел поскольку он требует больше 20-ти гигабайт дискового пространства для хранения полной копии blockchain. Синхронизация подобного узла с сетью занимает от трех дней, но это цена полной независимости и свободы от центральной власти.

Существует несколько альтернативных реализаций полного узла Bitcoin, построенные с использованием различных языков программирования и архитектур программного обеспечения. Тем не менее, наиболее распространенной реализацией выступает базовый клиент Bitcoin Core, также известный как клиент Satoshi. Более 90% узлов сети Bitcoin работают под управлением различных версий Bitcoin Core. В сообщении version, которое, как мы видели ранее, можно получить в ответ на команду getpeerinfo, подобные клиенты идентифицируются подстрокой "Satoshi", например, /Satoshi:0.8.6/.

Обмен "инвентарем"

Первое, что сделает полный узел, как только подключится к остальным пирам, это попытается построить полный blockchain. Если это совершенно новый узел, без blockchain вообще, он знает только о блоке генезиса, статически встроенном в клиентское ПО. Начиная с блока #0 (блока генезиса), новому узлу понадобится скачать сотни тысяч блоков для синхронизации с сетью и восстановить у себя blockchain полностью.

Процесс синхронизации с blockchain начинается с сообщения version, т.к. в нем содержится

BestHeight, текущую высоту blockchain (количества блоков) на узле. Узел будет видеть сообщения version своих пиров, знать сколько блоков каждый из них содержит, и иметь возможность сравнить с собственным blockchain. Пиримы примутся обмениваться сообщениями getblocks, содержащими хэш верхнего блока своей локальной копии blockchain. Один из пиров сможет идентифицировать полученный хэш как принадлежащий к блоку, который не находится на вершине, а скорее относится к старому блоку, тем самым сделав вывод, что его собственная локальная копия blockchain старше, чем у остальных пиров.

Узел, располагающий более длинной цепочкой блоков может определить, каких блоков не хватает другому узлу. Этот узел определит первые таких 500 блоков и передаст их хэши с помощью сообщения inv (инвентарь). Узел, у которого этих блоков не хватает, примет их затем после серии сообщений getdata, запрашивающих полные данные блоков и идентифицируя их с помощью хэшей из сообщения inv.

Предположим, например, что некий узел содержит только блок генезиса. Сперва-наперво он получит сообщение inv от других пиров, содержащее хэши следующих 500 блоков в цепи. Он начнет запрашивать блоки у всех своих подключенных пиров, распределяя нагрузку и обеспечивая, чтобы пиры не были перегружены запросами. Узел отслеживает, сколько блоков из запрошенных находятся "в пути" по каждому из соединений, проверяя, чтобы это количество не превышало предел (MAX_BLOCKS_IN_TRANSIT_PER_PEER). Таким образом, если узлу понадобится большее количество блоков, он будет запрашивать новые только как только выполнятся предыдущие запросы, что позволяет пирам контролировать темп обновлений и не перегружать сеть. По мере получения, каждый блок добавляется к blockchain, как мы увидим в [\[blockchain\]](#). По мере роста локального blockchain, все больше блоков запрашивается из сети, и процесс продолжается до тех пор, пока узел не догонит остальную часть сети.

Этот процесс сравнения локального blockchain с пирами и получения каких-либо недостающих блоков происходит каждый раз после того, как узел уходит в офлайн в течение любого периода времени. Если узел был автономном режиме в течение нескольких минут и ему не хватает нескольких блоков, или на месяц, и не хватает несколько тысяч блоков, она начинает с отправки getblocks и получения inv в ответ, а затем начинает загрузку недостающих блоков. В [Узел синхронизируется с блокчейном путем получения блоков от другого узла](#) показан протокол инвентаризации и распространения блоков.

Узлы упрощенного подтверждения оплаты (SPV)

Не все узлы способны хранить полную копию blockchain. Многие Bitcoin-клиенты предназначены для работы на в сильно ограниченных устройствах, таких как смартфоны, планшетах или встраиваемых системах. Для подобных устройств используется *simplified payment verification* (SPV), который позволяет им работать без локальной копии blockchain. Эти типы клиентов называются SPV-клиентами или облегченными клиентами. По мере роста распространения технологии Bitcoin, SPV-узлы становятся наиболее распространенной формой Bitcoin узла, особенно для Bitcoin кошельков.

SPV-узлы загружают только заголовки блоков и не загружают транзакции, включенные в

каждый блок. Полученная цепочка блоков, без транзакций, в 1000 раз меньше, чем полный blockchain. SPV-узлы не могут построить полную картину всех UTXO, доступных для расходов, т.к. они не знают обо всех транзакциях в сети. SPV-узлы верифицируют транзакции с использованием несколько иной методологии, которая опирается на пиры для обеспечения просмотра соответствующих частей blockchain по требованию.

Node A

Node B

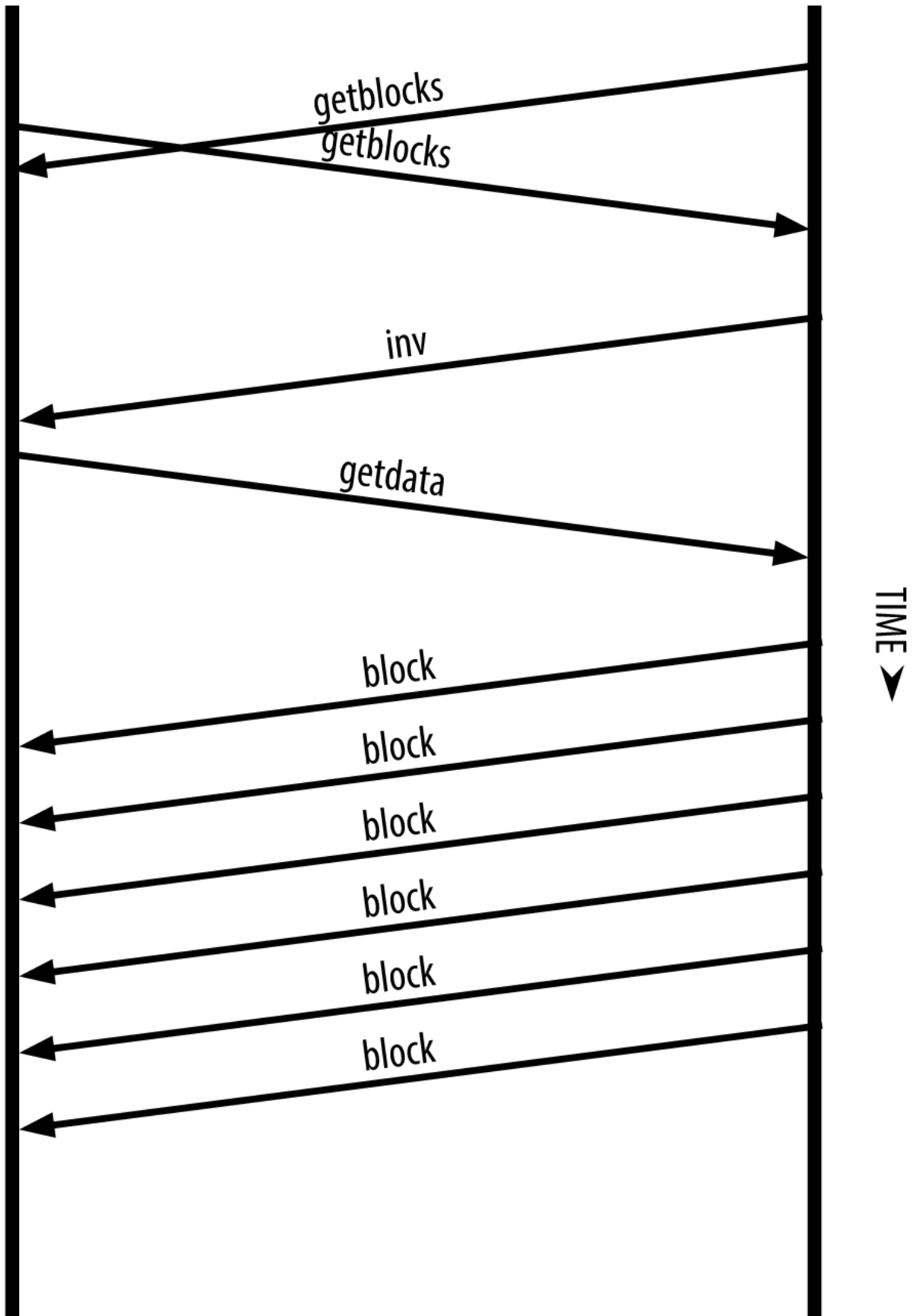


Figure 6. Узел синхронизируется с блокчейном путем получения блоков от другого узла

В качестве аналогии, полный узел—это турист в незнакомом городе, у которого есть подробная карта каждой улицы и каждого адреса. Напротив, SPV-узел—это турист в незнакомом городе, который знает только главную улицу и спрашивает случайных встречающих о том, как пройти дальше. Хотя оба туриста могут проверить существование улицы, просто посетив ее, турист без карты не знает о существовании других улиц. Находясь по адресу "23 Church Street", турист без карты не может знать о существовании десятка других "23 Church Street" адресов в городе и находится ли он сам по правильному адресу. Лучшее, что может сделать турист без карты, это опросить достаточное количество людей, и надеяться, что некоторые из них не попытаются его обмануть.

Упрощенная верификация платежа проверяет транзакции с помощью ссылки на их глубину в blockchain вместо *высоты*. В то время как полный узел будет полностью проверять цепочку тысяч блоков и транзакций вниз по blockchain (во времени) вплоть до блока генезиса, SPV-узел будет проверять цепь всех блоков (но не все транзакции) и связывать эту цепь с интересующей транзакцией.

Например, при рассмотрении транзакции в блоке 300000, полный узел прослеживает все 300000 блоков вниз до блока генезиса и строит полную базу данных UTXO, устанавливая валидность транзакций, подтверждая, что UTXO остается неизрасходованным. SPV-узел не может проверить, является ли UTXO неизрасходованным. Вместо этого SPV-узел установит связь между транзакцией и блоком, который ее содержит при помощи *пути Меркле* (см. [\[merkle_trees\]](#)). Затем SPV-узел ждет, пока не увидит шесть блоков с 300001 до 300006 сложенных поверх блока, содержащего транзакцию и проверяет его путем установления его глубины под блоками от 300006 до 300001. Тот факт, что другие узлы сети приняли блок 300000, а затем проделали необходимую работу по производству еще шести блоков поверх него является косвенным доказательством того, что транзакция не производит двойную трату.

SPV-узел не может быть уверен, что транзакция существует в блоке, когда транзакция и на самом деле не существует. SPV-узел устанавливает существование транзакции в блоке путем запроса доказательства пути Меркле и проверки доказательства работы в цепочке блоков. Однако, существование транзакции может быть "скрытым" от SPV-узла. SPV-узел может определенно доказать, что транзакция существует, но не может проверить, что транзакция повторно расходующая один и тот же UTXO, не существует, поскольку он не содержит записей всех транзакций. Эта уязвимость может быть использована для атак на отказ в обслуживании или для атак двойного расходования на SPV-узлы. Чтобы защититься от подобных атак, SPV-узел должен подключаться случайным образом к нескольким узлам, чтобы увеличить вероятность того, что по меньшей мере один из них честный. Эта потребность подключения к случайным узлам означает, что SPV-узлы также уязвимы для атак разделения сети или атак Сибилы, где они подключены к поддельными узлам или сетям и не имеют доступа к честным узлам или реальной сети Bitcoin.

Для большинства практических целей, узлы SPV достаточно безопасны, обеспечивая правильный баланс между потребностями в ресурсах, практичностью и безопасностью. В смысле абсолютной безопасности, однако, ничто не сравнится с полным узлом.

ТИР

Полный узел проверяет транзакцию отслеживая всю цепочку из тысяч блоков, чтобы гарантировать, что UTXO не потрачен, в то время как узел SPV проверяет как глубоко блок закопан с помощью горстки блоков над ним.

Чтобы получить заголовки блоков, SPV-узлы используют сообщение `getheaders` вместо `getblocks`. Отвечающий узел отправит до 2000 заголовков блоков в одном единственном сообщении `headers`. В остальном процесс такой же, как для полного узла при получении полных блоков. SPV-узлы также используют фильтры на подключениях к пирам для фильтрации потока будущих блоков и транзакций, посланных пирами. Любые транзакции, представляющие интерес извлекаются с помощью запроса `getdata`. Пир генерирует сообщение `tx`, содержащее транзакции в качестве ответа. В [SPV-узел синхронизирует заголовки блоков](#) показана синхронизация заголовков блоков.

Node A

Node B

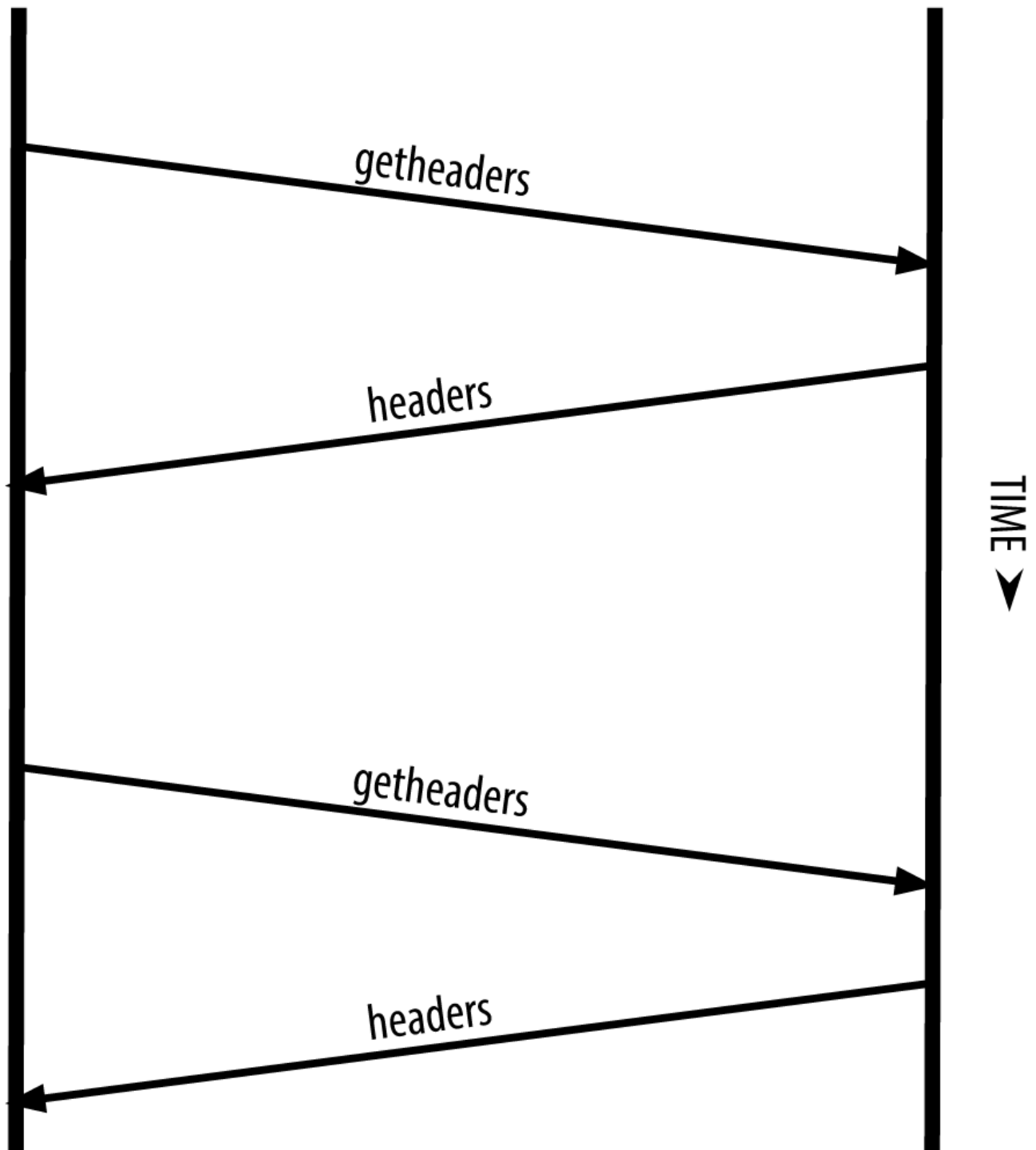


Figure 7. SPV-узел синхронизирует заголовки блоков

Поскольку SPV-узлы получают определенные транзакции для их выборочной проверки, они также создают угрозу для конфиденциальности. В отличие от полных узлов, которые собирают все транзакции внутри каждого блока, запросы определенных адресов SPV-узлами могут непреднамеренно проявить адреса их кошельков. Например, третья сторона, прослушивающая сеть может отслеживать все транзакции, запрашиваемые кошельком SPV-

узла и использовать эту информацию для связывания Bitcoin-адреса с пользователем этого кошелька, тем самым поставив под угрозу конфиденциальность пользователя.

Вскоре после введения SPV/легких узлов, разработчики Bitcoin добавили функцию под названием *фильтры Блума* для решения рисков конфиденциальности SPV-узлов. Фильтры Блума позволяют SPV-узлам получить подмножество транзакций не раскрывая, в каких именно адресах они заинтересованы, через механизм фильтрации, использующий вероятности, а не фиксированные шаблоны.

Фильтры Блума

Фильтр Блума представляет собой вероятностный фильтр поиска, способ для описания требуемого шаблона без его точного указания. Фильтры Блума обеспечивают эффективный способ выражения шаблона поиска с сохранением степени конфиденциальности. Эти фильтры используются SPV-узлами для опроса пиров о транзакциях, соответствующих определенному образцу, не раскрывая адреса поиска.

В нашей предыдущей аналогии, турист без карты спрашивает направление для конкретного адреса "23 Church St.". Если он спрашивает незнакомцев дорогу к этому месту, он неосторожно показывает пункт своего назначения. Фильтр Блума позволяет как бы спросить, "Есть ли какие-то улицы в этом районе, название которых оканчивается на R-C-H?". Вопрос проде этого открывает немного меньше информации о цели путешествия, чем вопрос о "23 Church St.". Используя подобную технику, турист может указать желаемый адрес чуть более подробно, например, "адрес заканчивается на U-R-C-H" или менее подробно, вроде "окончивается на H." Изменяя точность поиска, турист показывает информации больше или меньше, в замен получая больше или меньше конкретных ответов. Если он спрашивает менее конкретный шаблон, то получает гораздо больше возможных адресов и больше конфиденциальности, но многие результаты не будут иметь к запросу никакого отношения. Если он запросит более специфичный шаблон, то получит меньше результатов, но потеряет некоторую степень конфиденциальности.

Фильтры Блума служат этой функции, позволяя SPV-узлу задать шаблон поиска транзакций, предлагая выбор между точностью и конфиденциальностью. Более специфичный фильтр Блума будет давать более точные результаты, но за счет выявления адресов, использующихся кошельком пользователя. Менее специфический фильтр Блума будет давать больше данных о большем количестве транзакций, многие из которых не будут иметь отношения к узлу, но позволит узлу поддерживать лучшую степень конфиденциальности.

SPV-узел проинициализирует фильтр Блума как "пустой" и в таком состоянии фильтр Блума не будет соответствовать никакому шаблону. Далее SPV-узел составит список всех адресов своего кошелька и создаст шаблон поиска, соответствующий выходам транзакции для каждого из адресов. Как правило, шаблон поиска — это pay-to-public-key-hash сценарий ожидаемый запирающий скрипт, который будет присутствовать в любой транзакции на public-key-hash (адрес). Если же SPV-узел отслеживает баланс P2SH адреса, то поисковым шаблоном будет выступать сценарий pay-to-script-hash. Затем SPV-узел добавляет каждый из шаблонов поиска в

фильтр Блума, таким образом, чтобы фильтр смог распознать шаблон поиска, если он присутствует в транзакции. Наконец, фильтр Блума отправляется пиру и пир использует его для фильтрацией транзакций перед передачей на SPV-узел.

Фильтры Блума реализованы в виде массива переменного размера из N двоичных разрядов (битовых полей) и переменным числом хэш-функций M. Функции хэширования здесь всегда дают на выходе число от 1 до N, соответствующих двоичной матрице. Хеш-функции генерируются детерминировано, так что любой узел, реализующий фильтр Блума всегда будет использовать одни и те же хэш-функции и получит те же результаты для конкретного входа. Выбирая фильтры Блума различной длины (N) и различное число (M) хеш-функций, фильтр Блума можно настроить уровень точности и, следовательно, уровень конфиденциальности.

В [Пример упрощенного фильтра Блума, с 16-битовым полем и тремя хэш-функциями](#) мы используем очень маленький 16-ти битный массив и набор из трех хэш-функции, для демонстрации работы фильтров Блума.

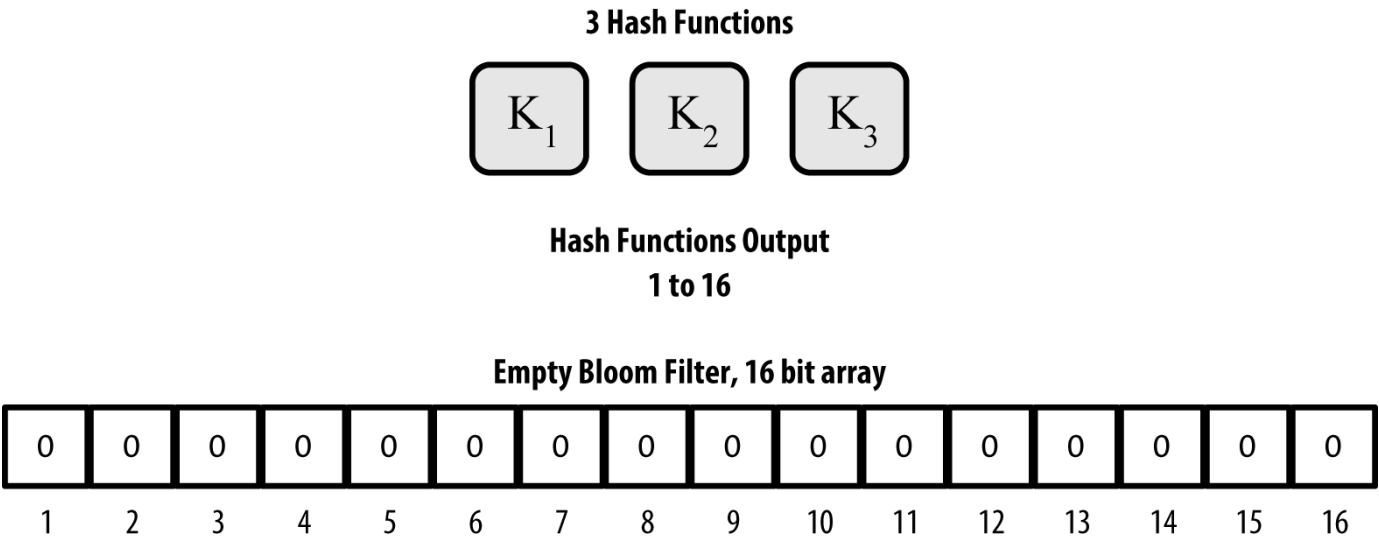


Figure 8. Пример упрощенного фильтра Блума, с 16-битовым полем и тремя хэш-функциями

Фильтр Блума инициализируется массивом, все биты которого установлены в 0. Для добавления шаблона к фильтру, он хешируется каждой функцией по очереди. Применение первой хеш-функции дает число между 1 и N. Соответствующий бит массива (пронумерованный по порядку от 1 до N) устанавливается в 1. Далее, следующая хеш-функция используется для установки еще одного бита и так далее. Как только все M хеш функций будет применены, поисковый шаблон будет "записан" в фильтре Блума в виде M битов, поменявших значение с 0 на 1.

[Добавление шаблона "А" к нашему простому фильтру Блума](#) пример добавления шаблона "А" к простому фильтру Блума, показанному на [Пример упрощенного фильтра Блума, с 16-битовым полем и тремя хэш-функциями](#).

Добавление второго шаблона так же просто, как повторять этот процесс. Шаблон хешируется каждой хеш-функцией по очереди, и результат записывается путем установки битов в 1.

Обратите внимание, что, по мере наполнения фильтра Блума шаблонами, результат хэш-функций может совпасть с битом, который уже был установлен в 1, причем в этом случае бит не изменяется. Больше шаблонов означает больше перекрывающихся битов установленных в 1, фильтр Блума насыщается и его точность уменьшается. Вот почему фильтр представляет собой вероятностную структуру данных — он становится менее точным по мере добавления новых моделей. Точность зависит от количества добавленных шаблонов, размера битового массива (N) и количества хэш-функций (M). Большой битовый массив и больше хеш-функций помогут запомнить больше шаблонов с более высокой точностью. Меньший битовый массив или меньшее количество хэш-функции дадут меньшую точность.

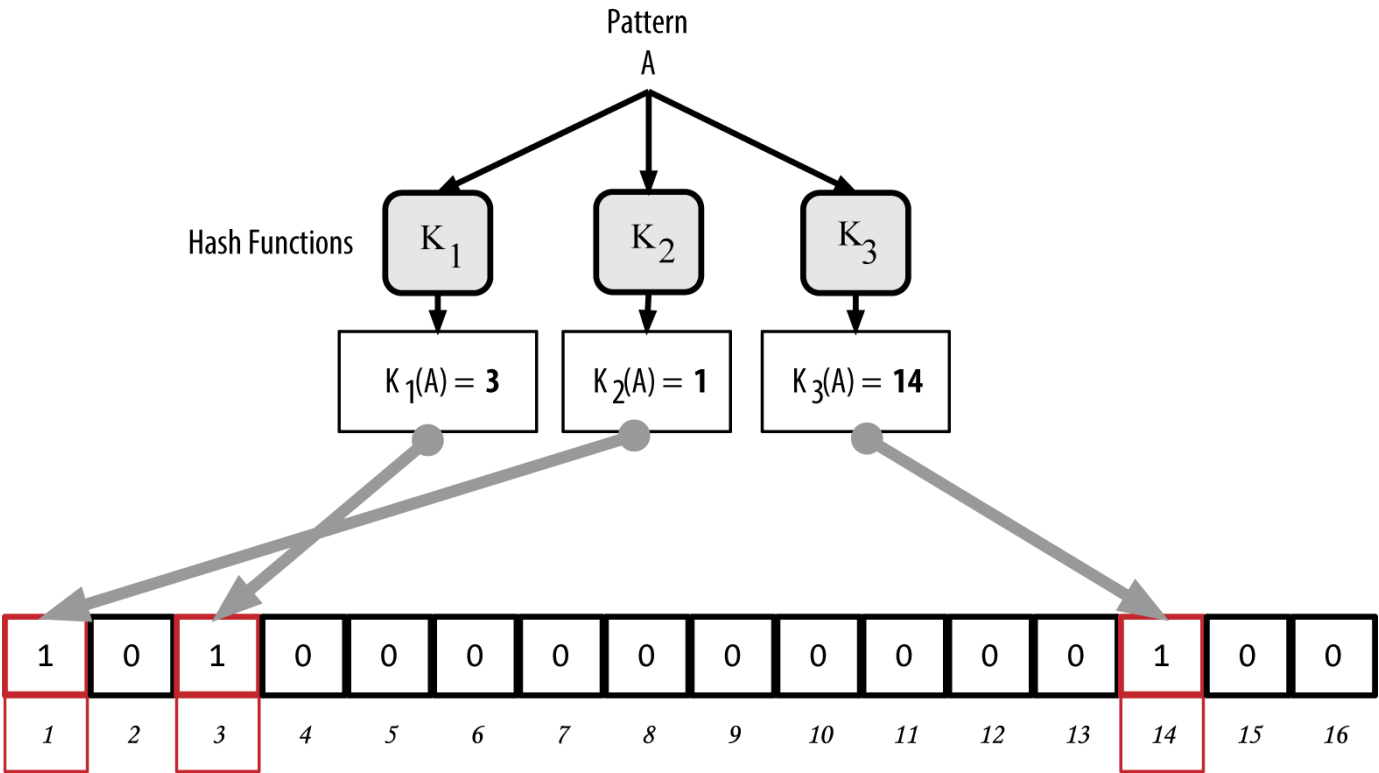


Figure 9. Добавление шаблона "А" к нашему простому фильтру Блума

Добавление второго шаблона "В" к нашему простому фильтру Блума пример добавления второго шаблона "В" к простому фильтру Блума.

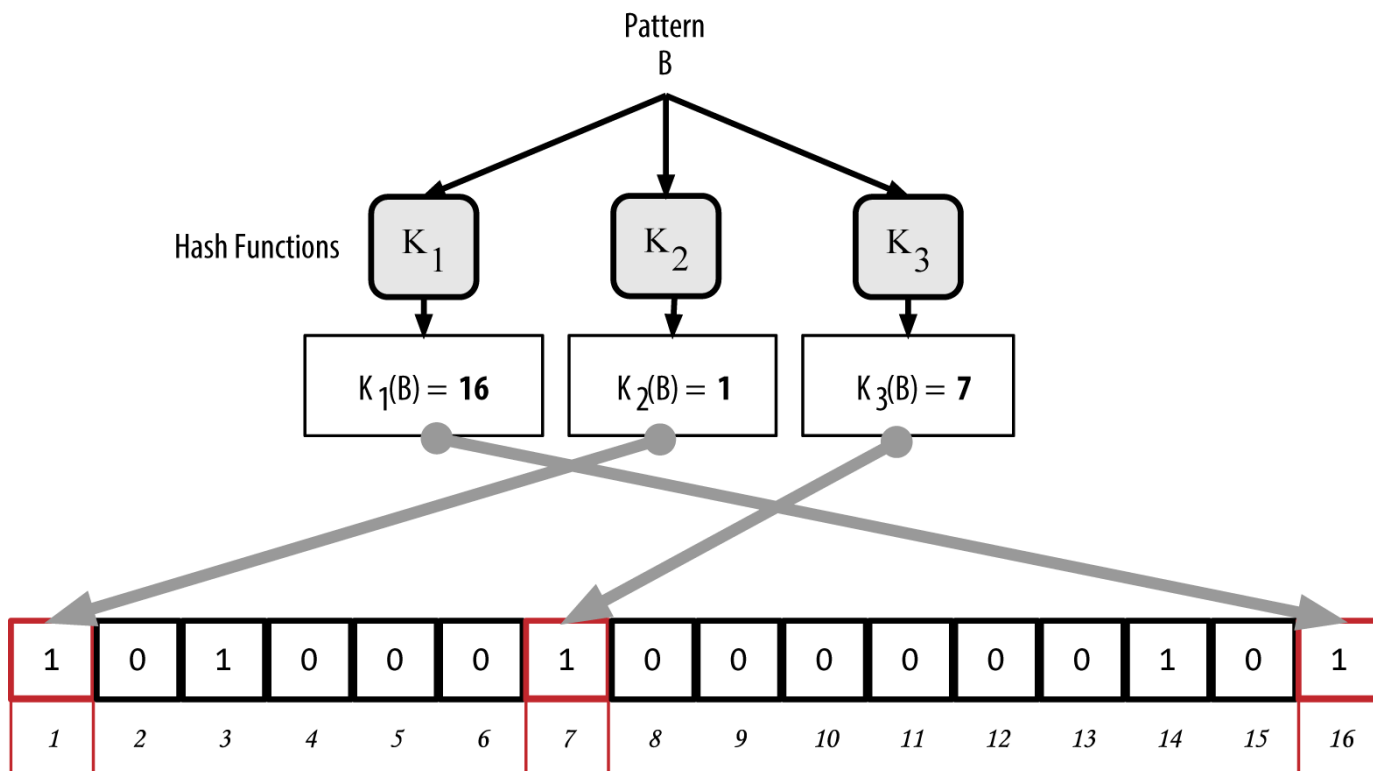


Figure 10. Добавление второго шаблона "B" к нашему простому фильтру Блума

Чтобы проверить, является ли шаблон частью фильтра Блума, шаблон хешируется каждой хэш-функцией и полученный битовый шаблон проверяется на соответствие битовому массиву. Если все биты индексированные хеш-функциями установлены в 1, то шаблон *вероятно* записан в фильтре Блума. Поскольку биты могут быть установлены в результате перекрытия нескольких шаблонов, ответ не точный, а скорее вероятностный. Говоря простыми словами, положительный результат на выходе фильтра Блума означает "Может быть да."

Проверка существования шаблона "X" в фильтре Блума. Результат *вероятностно положительное совпадение, что означает «может быть»*. пример проверки существования шаблона "X" в простом фильтре Блума. Соответствующие биты устанавливаются в 1, поэтому шаблон вероятно найден.

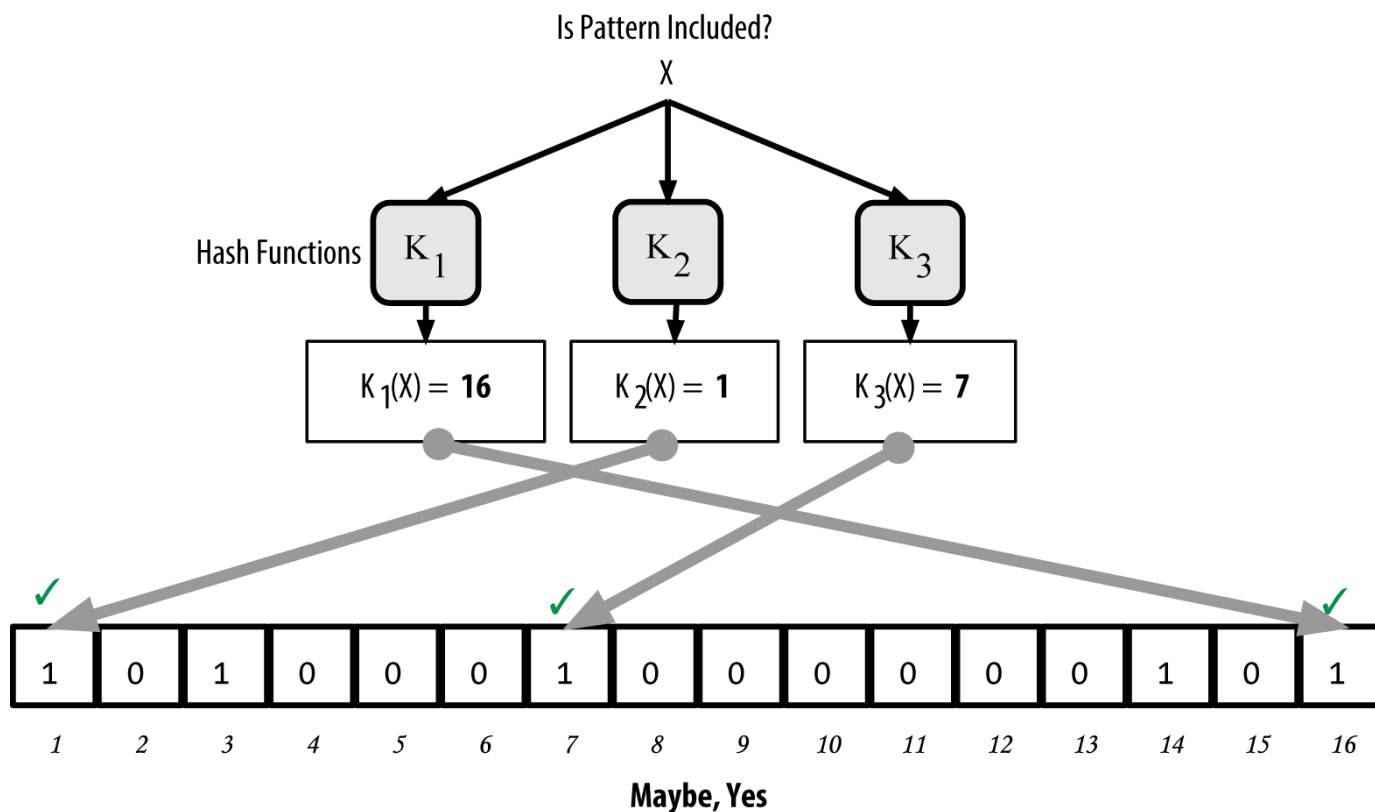


Figure 11. Проверка существования шаблона "X" в фильтре Блума. Результат вероятно положительное совпадение, что означает «может быть».

Наоборот, если шаблон проверяется на соответствие цвету фильтра и любой из битов установлен в 0, это означает, что шаблон не был записан в фильтр Блума. Отрицательный результат не вероятностный, а фактический. Говоря простыми словами, отрицательный результат на выходе фильтра Блума означает "Конечно, нет!"

В Проверка существования шаблона "Y" в фильтре Блума. Отрицательный результат означает "Конечно, нет!" показан пример проверки существования шаблона "Y" в простом фильтре Блума. Один из соответствующих битов устанавливается в 0, так что шаблон, безусловно, не найден.

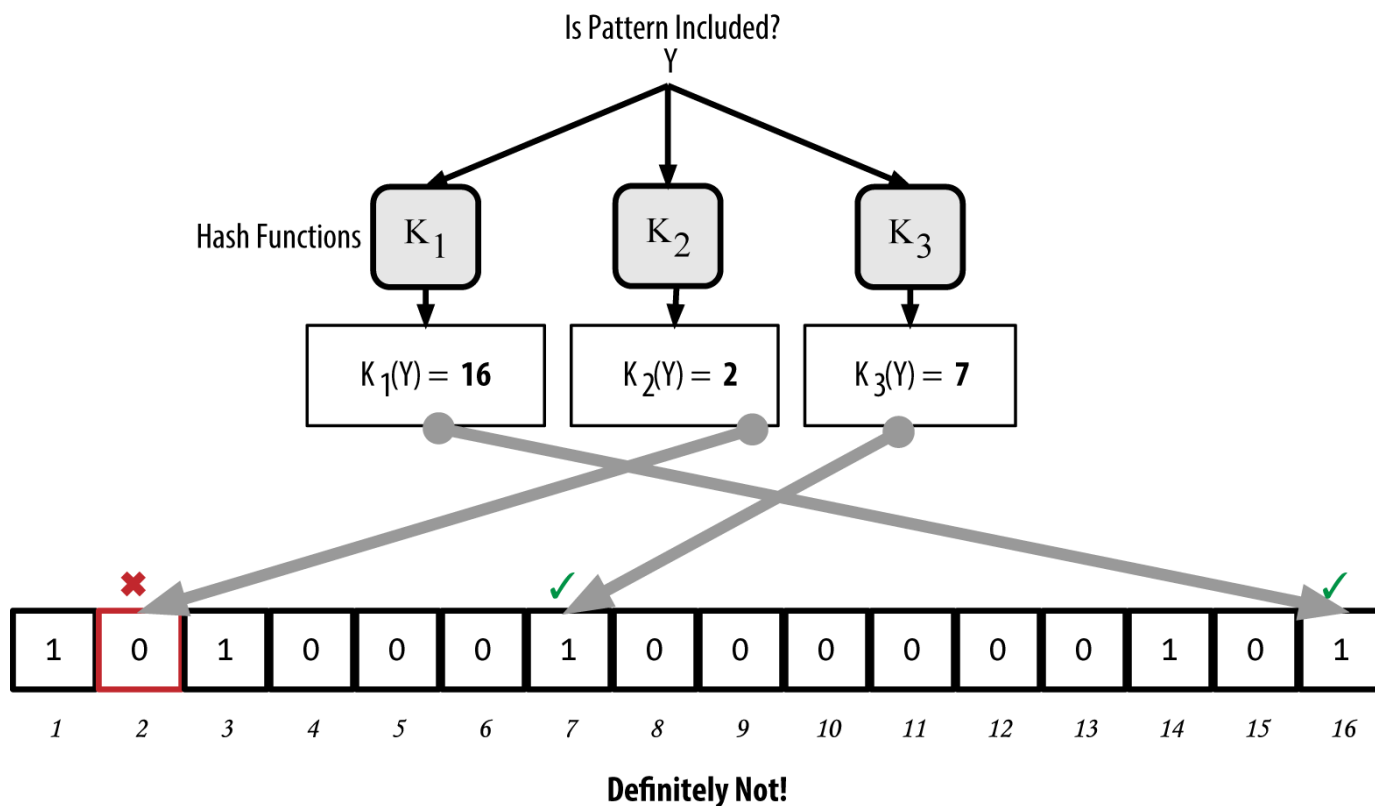


Figure 12. Проверка существования шаблона "Y" в фильтре Блума. Отрицательный результат означает "Конечно, нет!"

Реализация фильтра Блума в Bitcoin описана в Bitcoin Improvement Proposal 37 (BIP0037). См [\[appdxbitcoinimpproposals\]](#) или посетите [GitHub](#).

Фильтр Блума и обновления инвентаря

Фильтры Блума используются для фильтрации транзакций (и блоков, содержащих их), которые SPV-узел получает от своих пиров. SPV-узлы создают фильтр только для адресов кошелька SPV-узла. SPV-узел отправляет сообщение `filterload` содержащее фильтр своему пиру для использования в текущем соединении. После того, как фильтр установлен, пир будет проверять выходы каждой транзакции относительно фильтра и только транзакции, прошедшие фильтр, будут посланы узлу.

В ответ на сообщение `getdata`, пиры будут посылать сообщение `merkleblock`, содержащее только заголовки блоков для блоков, удовлетворяющих фильтру и пути Меркле (см. [\[merkle_trees\]](#)) для каждой удовлетворяющей транзакции. Пир будет также посылать сообщения `tx`, содержащие транзакции, удовлетворяющие фильтру.

Узел, устанавливающий фильтр Блума может интерактивно добавлять шаблоны к фильтру, посылая сообщение `filteradd`. Чтобы очистить фильтр Блума, узел может отправить сообщение `filterclear`. Так как удалить шаблон из фильтра Блума нельзя, узел должен очистить фильтр и повторно послать новый фильтр Блума, если шаблон больше не требуется.

Пулы транзакций

Почти каждый узел сети Bitcoin поддерживает временный список неподтвержденных транзакций, который называется *memory pool*, *mempool* или *пул транзакций* (*transaction pool*). Узлы используют этот пул, чтобы отслеживать транзакции, которые известны сети, но еще не включены в blockchain. Например, узел кошелька пользователя будет использовать пул транзакций для отслеживания входящих платежей, которые были получены сетью, но еще не были подтверждены.

По мере поступления и проверки транзакций, они добавляются в пул транзакций и передаются на соседние узлы в целях дальнейшего распространения по сети.

В некоторых реализациях узлов также поддерживается отдельный пул транзакций-сирот. Если входы транзакции ссылаются на пока еще не известную транзакцию, например, отсутствует родитель, транзакция-сирота будет временно храниться в таком пуле, пока не поступит родительская транзакция.

Когда транзакция добавляется в пул транзакций, сиротский пул проверяется на наличие каких-либо сирот, которые ссылаются на выходы этой транзакции (ее детей). Любые соответствующие сироты затем валидируются. Если валидация успешна, они удаляются из сиротского пула и добавляются в пул транзакций, завершая цепь, которая началась с родительской транзакции. В свете вновь добавленной сделки, которая уже не сирота, процесс повторяется рекурсивно в поисках каких-либо дальнейших потомков, пока не будут найдены все потомки. Благодаря этому процессу, прибытие родительской транзакции запускает каскад реконструкции целой цепочки взаимосвязанных транзакций при помощи пересоединения сирот со своими родителями на всем пути вниз по цепочке.

И пул транзакций и пул бесхозных транзакций (где реализован) хранятся в локальной памяти и не сохраняются на постоянных накопителях, скорее, они динамически заполняются из входящих сетевых сообщений. При запуске узла, оба пула пусты и постепенно заполняются новыми транзакциями, получаемыми из сети.

Некоторые реализации Bitcoin-клиента также поддерживают базу данных UTXO или UTXO-пул, который представляет собой множество всех неизрасходованных выходов в blockchain. Хотя название "UTXO пул" звучит похожим на пул транзакций, он представляет собой другой набор данных. В отличие от пула транзакций и сиротского пула, UTXO-пул не инициализируется пустым, а, наоборот, содержит миллионы записей неизрасходованных результатов транзакций, в том числе некоторые из которых относятся к 2009 г. UTXO-пул может быть размещен в локальной памяти или в качестве индексированной таблицы в базе данных на постоянном носителе.

В то время как пул транзакций и сиротские пулы представляют собой местную перспективу единственного узла и могут существенно отличаться от узла к узлу в зависимости от того, когда узел был запущен или перезагружен, UTXO-пул представляет собой возникающий консенсус сети и, следовательно, будет слегка меняться от узла к узлу. Кроме того, транзакции

и сиротские пулы содержат только неподтвержденные транзакции, в то время как UTXO-пул содержит только подтвержденные выходы.

Предупреждающие сообщения

Предупреждающие сообщения—это редко используемая функция, но, тем не менее реализованная в большинстве узлов. Предупреждающие сообщения—это "аварийная широковещательная система", с помощью которой разработчики ядра Bitcoin могут отправить аварийную текстовое сообщение всем узлам Bitcoin. Эта возможность позволяет главным разработчикам уведомить всех Bitcoin пользователей о какой-либо серьезной проблеме в сети Bitcoin, такой, например, как критическая ошибка, которая требует вмешательства пользователя. Система оповещения использовалась только несколько раз, особенно в начале 2013 года, когда критическая ошибка базы данных вызвала возникновение многоблокового форка в blockchain.

Предупреждающие сообщения распространяются при помощи сообщения alert. Предупреждающее сообщение содержит несколько полей, в том числе:

ID

Идентификатор предупреждения требуемый, чтобы могли быть обнаружены дублирующие сообщения

Expiration

Время, после которого предупреждение считается истекшим

RelayUntil

Время, после которого предупреждение не должно дальше распространяться по сети

MinVer, MaxVer

Диапазон версий протокола Биткоин, к которым относится это предупреждение

subVer

Версия клиентского программного обеспечения, к которому относится это предупреждение

Priority

Уровень приоритета предупреждения, в настоящее время не используется

Предупреждающие сообщения криптографически подписаны публичным ключом. Соответствующий приватный ключ находится у нескольких избранных членов команды разработчиков ядра. Цифровая подпись гарантирует, что по сети не смогу быть посланы фальшивые сообщения.

Каждый узел, принявший данное предупреждение проверит его, проверит срок действия, и распространит его дальше всем своим пирам, тем самым обеспечивая быстрое распространение по всей сети. В дополнение к распространению оповещения, узлы могут

показать уведомление пользователю.

В клиенте Bitcoin Core, оповещения настраивается параметром командной строки `-alertnotify`, который определяет команду для запуска при получении предупреждения. Предупреждающее сообщение передается в качестве параметра команды `alertnotify`. Чаще всего команда `alertnotify` устанавливается на отправку электронной почты с предупреждением администратору узла. Предупреждение также отображается во всплывающем окне в графическом интерфейсе пользователя (Bitcoin-Qt), если он запущен.

Другие реализации протокола Bitcoin могут обрабатывать оповещение различными способами. Многие аппаратные майнинговые Bitcoin-системы не реализуют функцию сообщения оповещения, потому что они не имеют никакого пользовательского интерфейса. Настоятельно рекомендуется, чтобы майнеры, использующие подобные системы, подписались на оповещения от оператора пула или запустив легкий узел только в целях оповещения.

Блокчейн

Введение

Структура данных blockchain — это упорядоченный обратно связанный список блоков транзакций. Blockchain может храниться в плоском файле или в простой базе данных. Клиент Bitcoin Core хранит метаданные blockchain используя БД LevelDB от Google. Блоки связаны "назад", каждый ссылается на предыдущий блок в цепи. Blockchain часто визуализируется как вертикальная стопка, с блоками, лежащими друг на друге и первым блоком, служащим основанием. Подобная визуализация в виде блоков, сложенных друг на друга приводит к использованию таких терминов, как "высота" для обозначения расстояния от первого блока, и "вершина", указывающая на недавно добавленный блок.

Каждый блок в blockchain идентифицируется хешем, который генерируется с использованием криптографического алгоритма SHA256, примененного к заголовку блока. Каждый блок также ссылается на предыдущий блок, известный как *родительский* блок, через поле "хэш предыдущего блока" в заголовке блока. Другими словами, каждый блок содержит хеш своего родителя внутри собственного заголовка. Последовательность хэшей, связывающих каждый блок с его родителем создает цепь, тянущуюся к самому первому блоку из когда-либо созданных, известному как *блок генезиса*.

Хотя блок имеет только одного родителя, он может временно иметь несколько дочерних блоков. Каждый из дочерних блоков ссылается на один и тот же родительский блок и содержит тот же хэш в поле "хэш предыдущего блока". Несколько дочерних блоков возникают во время "разветвления" blockchain, временной ситуации, которая возникает, когда несколько разных блоков обнаруживаются почти одновременно разными майнерами (см [\[forks\]](#)). В конце концов, только один дочерний блок становится частью blockchain и "разветвление" пропадает. Даже если блок может иметь более одного дочернего, каждый блок может иметь только один родительский блок. Это потому, что каждый блок имеет одно единственное поле "хэш предыдущего блока", ссылающееся на одного родителя.

Поле "хэш предыдущего блока" внутри заголовка блока и тем самым влияет на *текущий* хэш блока. Хэш дочернего блока меняется, если меняется хэш родительского. Когда родительский блок получает какие-либо изменения, меняется его хэш. Измененный хэш родительского блока требует изменения указателя "хэша предыдущего блока" в дочернем блоке. Это в свою очередь изменяет хэш самого дочернего блока, которое, в свою очередь, изменяет указатель во внучатом блоке, который, в свою очередь изменяет хэш внучатого блока, и так далее. Это каскадный эффект гарантирует, что если за блоком последовало много поколений, он не может быть изменен без пересчета всех последующих блоков. Так как подобный пересчет потребует огромного количества вычислений, существование длинной цепи блоков делает глубокую историю в блокчейне неизменяемой, что является ключом к безопасности Биткоин.

Blockchain можно представить в виде слоистой геологической формации, или ледника. Поверхностные слои могут изменяться со временем, или даже быть сдуты ветром, прежде, чем

они успевали бы осесть. Но, если копнуть на несколько дюймов вглубь, геологические слои становятся все более и более стабильными. А если копнуть на несколько сотен футов вниз, вы увидите снимок прошлого, нетронутого в течение миллионов лет. В blockchain, самые последние несколько блоков могут быть пересмотрены, если произойдет перерасчет в связи с разветвлением. Верхние шесть блоков — это как несколько дюймов верхнего слоя почвы. Но как только вы погружаетесь глубже в blockchain, больше, чем на шесть блоков, вероятность возникновения изменений все меньше и меньше. На расстоянии 100 блоков от верхушки, стабильность настолько гарантирована, что майнерам разрешаются транзакции со средствами из coinbase новых блоков. Несколько тысяч блоков назад (месяц) и blockchain — устоявшаяся история для всех практических целей. Хотя протокол всегда позволяет отменить цепоску с помощью более длинной цепочки и хотя всегда существует вероятность отмены блока, вероятность такого события снижается с течением времени, пока он не станет бесконечно малой.

Структура блока

Блок представляет собой структуру данных, контейнер, объединяющий транзакции для включения в общедоступную бухгалтерскую книгу, в blockchain. Блок состоит из заголовка, содержащего метаданные, последующего за ним длинного списка транзакций, которые занимают большую часть всего размера блока. Заголовок блока занимает 80 байт, в то время как средняя транзакция занимает не менее 250 байт, а средний блок содержит более 500 сделок. Соответственно, полностью заполненный транзакциями блок в 1000 раз больше заголовка. [Структура блока](#) описывает структуру блока.

Table 1. Структура блока

Размер	Поле	Описание
4 байта	Размер блока	Размер блока в байтах
80 байт	Заголовок блока	Несколько полей образуют заголовок блока
1-9 байт (VarInt)	Счетчик транзакций	Количество транзакций в блоке
Переменная	Транзакции	Транзакции, записанные в этом блоке

Заголовок блока

Заголовок блока состоит из трех наборов метаданных. Во-первых, там содержится хеш предыдущего блока. Второй набор метаданных, а именно сложность, временная метка и некое случайное число (nonce), заполняемые майнерами, как указано в [ch8]. Третья часть метаданных содержит корень дерева Меркле, структуры данных, используемой для эффективного объединения всех транзакций в блоке. В [Структура заголовка блока](#) подробно описана структура заголовка блока.

Table 2. Структура заголовка блока

Размер	Поле	Описание
4 байта	Версия	Номер версии для отслеживания обновлений программного обеспечения/протоколов
32 байта	Хэш Предыдущего Блока	Ссылка на хэш предыдущего (родительского) блока в цепочке
32 байта	Корень дерева Меркле	Хэш корня дерева Меркле транзакций этого блока
4 байта	Временная метка	Приблизительное время создания этого блока (секунды от эпохи Unix)
4 байта	Целевая сложность	Целевая сложность алгоритма доказательства работы для данного блока
4 байта	Некое случайное число (nonce)	Счетчик используется в алгоритме доказательства работы

Случайное число, целевая сложность, и временная метка используются в майнинге и будут обсуждаться более подробно в [\[ch8\]](#).

Идентификаторы блока: хеш заголовка блока и высота блока

Основной идентификатор блока — это его криптографической хэш, цифровой отпечаток, полученный при хэшировании заголовка блока дважды алгоритмом SHA256. Результирующий 32-байтовый хэш называется *хешем блока*, но более правильно его называть *хешем заголовка блока*, *потому что для его вычисления используется только заголовок блока, например* `000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26fÉÔÉ` это хеш самого первого блока Биткоин из когда-либо созданных. Хеш блока однозначно и недвусмысленно идентифицирует блок и может быть получен независимо любым узлом просто в результате хэширования заголовка блока.

Следует отметить, что хеш блока фактически не включен в структуру данных блока, ни когда блок передается по сети, ни когда он хранится в базе данных узлов, как часть blockchain. Вместо этого, хеш блока вычисляется каждым узлом, как только блок получен из сети. Хеш блока может храниться в отдельной таблице базы данных, как часть метаданных блока для

того, чтобы облегчить индексацию и ускорить считывание блоков с диска.

Второй способ идентификации блока — это по его положению в blockchain, называемом *высотой блока*. Самый первый блок имел высоту 0 (нуль) — это тот же блок, который мы ранее упоминали по его хешу 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. Блок может таким образом быть идентифицирован двумя способами: с помощью ссылки на хеш блока или с помощью высоты блока. Каждый последующий блок добавляется на одну позицию "выше" в blockchain. Высота блока на 1 января 2014 года, составляла приблизительно 278000, то есть с момента создания первого блока в январе 2009 года появилось 278000 блоков.

В отличие от хэша блока, высота блока не является уникальным идентификатором. Хотя один блок всегда будет иметь определенную неизменную высоту, обратное не верно — высота блока не всегда определяет единственный блок. Два или более блока могут иметь одинаковую высоту блока, конкурируя за одну и ту же позицию в blockchain. Этот сценарий подробно обсуждается в разделе [\[forks\]](#). Высота блока также не является частью структуры данных блока; она не хранится в блоке. Каждый узел динамически определяет положение (высоту) блока в blockchain, как только он получает блок из Биткойн-сети. Высота блока также может храниться в виде метаданных в индексированной таблице базы данных для быстрого поиска.

ТИП

Хеш блока всегда однозначно идентифицирует один блок. Блок также всегда имеет определенную *высоту блока*. Тем не менее не всегда определенная высота блока может идентифицировать единственный блок. Скорее, два или более блока могут конкурировать за одну позицию в blockchain.

Блок Генезиса

Первый блок в blockchain называется блоком генезиса. Он был создан в 2009 году и является общим предком всех блоков в blockchain. Это означает, что если вы выберете любой блок и проследите цепочку назад во времени, то в конечном итоге придете к блоку генезиса.

Каждый узел всегда начинается с blockchain по меньшей мере из одного блока так, как блок генезиса закодирован статически в клиентском ПО, что значит он не может быть модифицирован. Каждый узел всегда "знает" хэш блока генезиса и его структуру, фиксированное время, когда он был создан, и даже единственную транзакцию в этом блоке. Таким образом, каждый узел имеет отправную точку для blockchain, безопасный "корень", из которого можно построить доверительный blockchain.

Посмотреть статически закодированный генезис блок внутри клиента Bitcoin Core можно здесь [chainparams.cpp](#).

Блоку генезиса соответствует следующий идентифицирующий хэш:

```
00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Вы можете искать этот хеш блока в любом "проводнике по блокчейну" таком, как, например, blockchain.info, и найдете страницу, описывающую содержание этого блока, с URL, содержащим этот хэш:

<https://blockchain.info/block/00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

<https://blockexplorer.com/block/00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

Используя базовый клиент Bitcoin Core из командной строки:

```
$ bitcoind getblock 00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

```
{
  "hash" : "00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
  "confirmations" : 308321,
  "size" : 285,
  "height" : 0,
  "version" : 1,
  "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
  "tx" : [
    "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
  ],
  "time" : 1231006505,
  "nonce" : 2083236893,
  "bits" : "1d00ffff",
  "difficulty" : 1.00000000,
  "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

Блок генезиса содержит скрытое сообщение внутри себя. Вход coinbase-транзакции содержит текст "Таймс 03/Янв/2009 Министр экономики на грани второго раунда спасения банков." Это сообщение было предназначено в качестве доказательства самой ранней даты, когда этот блок мог быть создан, с помощью ссылки на заголовок британской газеты *The Times*. Он также служит напоминанием важности создания независимой монетарной системы, с запуска Биткоин в то же самое время, когда во всем мире бушует беспрецедентный валютный кризис. Сообщение было заложено в первый блок Сатоши Накамото, создателем Биткоин.

Связывание блоков в Blockchain

Полные узлы содержат локальную копию blockchain, начиная с блока генезиса. Локальная

копия blockchain постоянно обновляется и удлиняется по мере появления новых блоков. Как только узел принимает входящие блоки из сети, он проверяет их, а затем связывает с существующей цепочкой блоков на основании "хеша предыдущего блока".

Предположим, например, что узел содержит 277,314 блоков в локальной копии blockchain. Последний блок, о котором узел знает—это 277,314 имеет хеш заголовка 00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249.

Затем Биткоин-узел получает новый блок из сети, который он разберет следующим образом:

```
{
  "size" : 43560,
  "version" : 2,
  "previousblockhash" :
    "00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
  "merkleroot" :
    "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
  "time" : 1388185038,
  "difficulty" : 1180923195.25802612,
  "nonce" : 4215469401,
  "tx" : [
    "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",

    # [... пропущено много других транзакций ...]

    "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
  ]
}
```

Узел смотрит в содержимое поля previousblockhash этого нового блока, в котором содержится хеш родительского блока. Этот хеш уже известен узлу как хеш последнего блока в цепочке на высоте 277314. Таким образом, этот новый блок является дочерним последнего блока в цепочке и расширяет текущий blockchain. Узел добавляет этот новый блок в конец цепочки, что делает blockchain длиннее с новой высотой 277315. В [Блоки связаны в цепочку при помощи ссылок на хеш заголовка предыдущего блока](#) показана цепочка из трех блоков, соединенных указателями в поле previousblockhash.

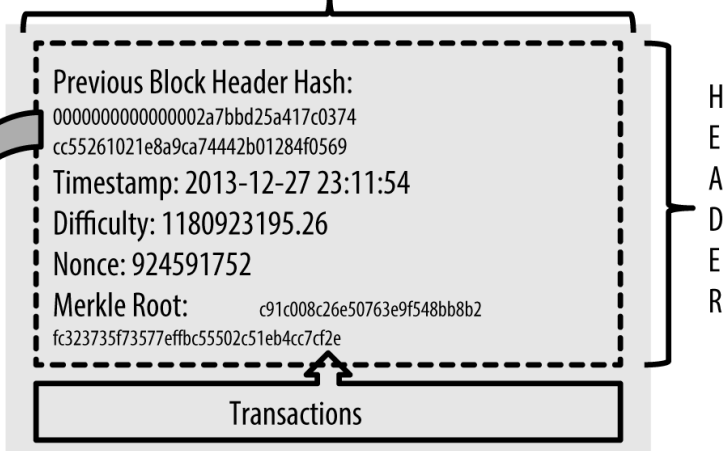
Деревья Меркле

Каждый блок в blockchain содержит "сводку" всех транзакций блока, используя *дерево Меркле*.

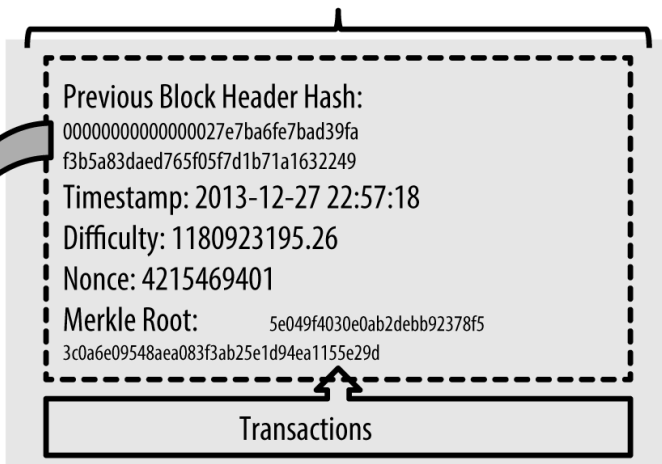
Дерево Меркле, также известное как *бинарное дерево хешей*—это структура данных, используемая для эффективного обобщения и проверки целостности больших наборов данных. Деревья Меркле—это бинарные деревья, содержащие криптографические хэши. Термин «дерево» используется в информатике для описания разветвленной структуры данных,

но эти деревья обычно отображается вверх ногами с "корнем" вверху и "листьями" в нижней части диаграммы, как показано ниже.

Block Height 277316
Header Hash:
000000000000001b6b9a13b095e96db
41c4a928b97ef2d944a9b31b2cc7bdc4



Block Height 277315
Header Hash:
000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569



Block Height 277314
Header Hash:
0000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249

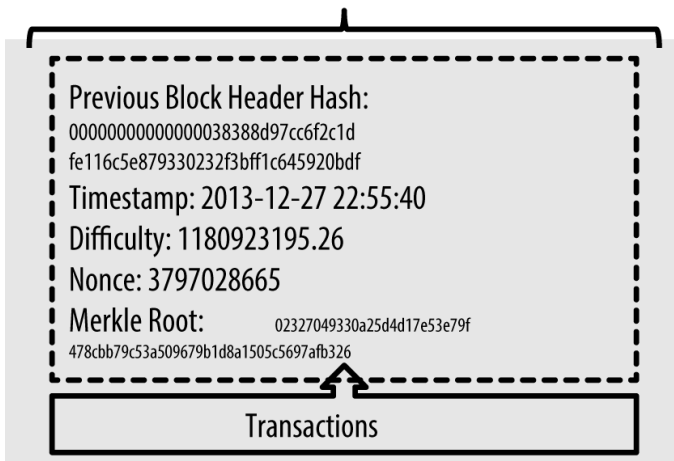


Figure 1. Блоки связаны в цепочку при помощи ссылок на хеш заголовка предыдущего блока

Деревья Меркле используются в Bitcoin для обобщения всех транзакций в блоке, производя общий цифровой отпечаток всей совокупности транзакций. С помощью этой структуры данных можно эффективно проверить нахождение транзакции в блоке. Дерево Меркле строится путем рекурсивного хэширования пар узлов до тех пор, пока не останется только один хэш, называемый *корнем* или *корнем меркле* (merkle root). В качестве криптографического хэша, используемом в деревьях Меркле Биткойна выступает двойной SHA256 (SHA256 применяется дважды).

Когда N элементов данных хэшируются и обобщаются деревом Меркле, проверка существования элемента в дереве потребует не более $2 \cdot \log_2(N)$ операций, делая эту структуру данных очень эффективной.

Дерево Меркле строится снизу вверх. В следующем примере, мы начинаем с четырех транзакций, A, B, C и D, которые формируют *листья* дерева Меркле, как показано на [Вычисление узлов дерева Меркле](#). Транзакции не хранятся в дереве Меркле; скорее, их данные хэшируются и полученный хеш хранится в каждом конечном узле как H_A , H_B , H_C , and H_D :

$$H_{A\sim} = \text{SHA256}(\text{SHA256}(\text{Транзакция A}))$$

Последовательные пары конечных узлов, которые затем сводятся в родительском узле путем конкатенации двух хэшей и хэширования результирующей строки. Например, чтобы построить родительский узел H_{AB} , два дочерних 32-байтовых хэша конкатинируются в 64-байтовую строку. Эта строка затем дважды хэшируется и получается хеш родительского узла:

$$H_{AB\sim} = \text{SHA256}(\text{SHA256}(H_{A\sim} + H_{B\sim}))$$

Процесс продолжается до тех пор, пока не остается только один узел, известный как *корень* Меркле. Этот 32-байтовый хэш хранится в заголовке блока и обобщает все данные во всех четырех транзакциях.

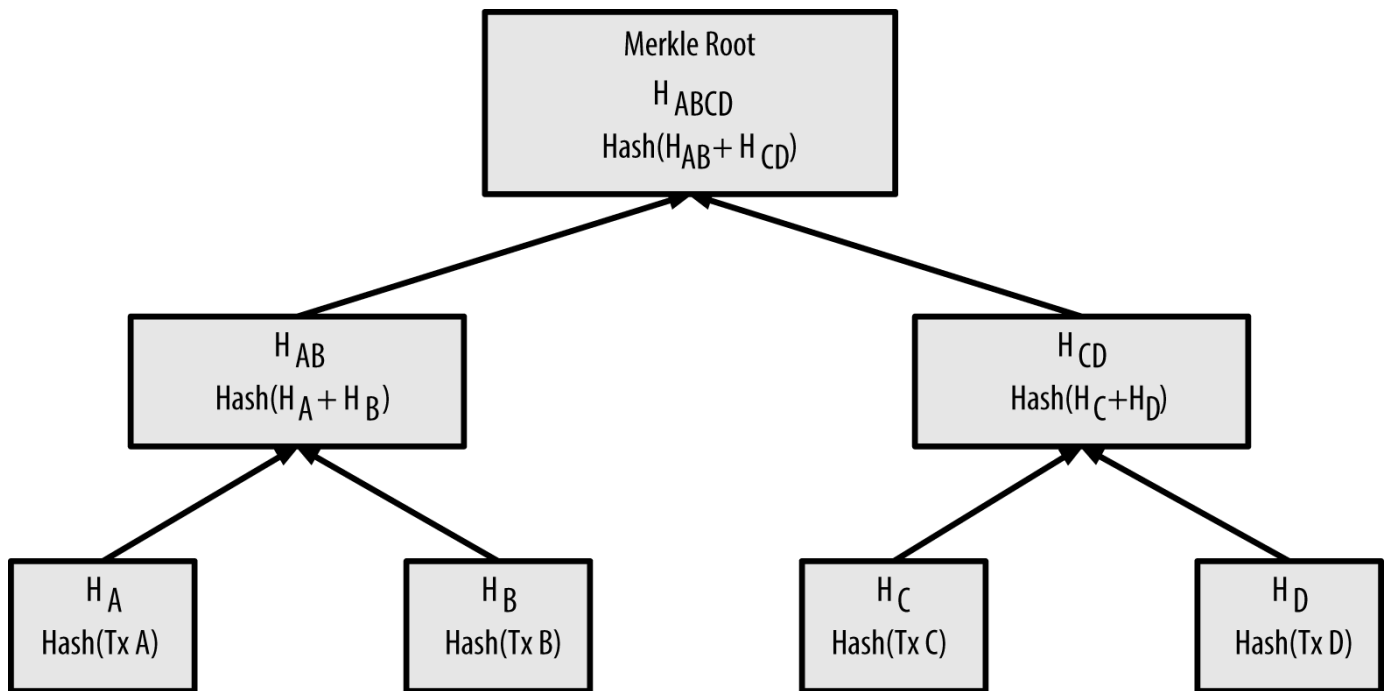


Figure 2. Вычисление узлов дерева Меркле

Поскольку дерево Меркле — это бинарное дерево, в нем должно быть четное количество листовых вершин. Если имеется нечетное количество транзакций, то хэш последней транзакции будет продублирован, последний хэш сделка будет дублироваться для получения так называемого *сбалансированного дерева*. Это показано на [Четное число элементов достигается с помощью дублирования одного элемента данных](#), где транзакция С дублируется.

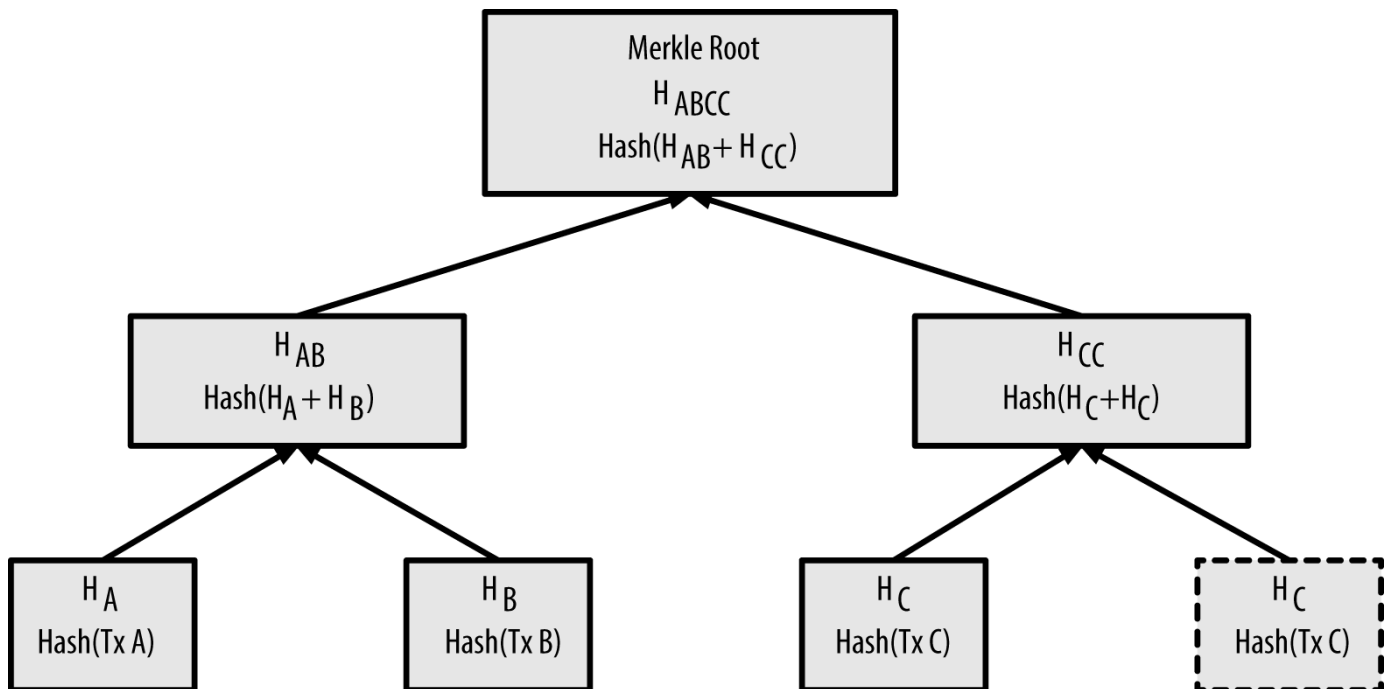


Figure 3. Четное число элементов достигается с помощью дублирования одного элемента данных

Тот же метод построения дерева из четырех транзакций можно обобщить до деревьев любого

размера. В одном блоке Bitcoin как правило несколько сотен транзакций, которые точно таким же образом обобщены до 32 байт данных в корне Меркле. В [Дерево Меркле, обобщающее множество элементов данных](#) показано дерево из 16-ти транзакций. Отметим, что хотя на схеме корень выглядит крупнее, чем листовые вершины, он имеет точно такой же размер, всего 32 байт. Не смотря на то, сколько транзакций в блоке: одна или сотни тысяч, корень Меркле всегда обобщает их до 32 байт.

Для поиска доказательства, что конкретная транзакция содержится в блоке, узлу требуется произвести всего лишь $\log_2(N)$ 32-байтовых хешей, составляя *аутентификационный путь* или *путь Меркле*, соединяющий конкретную транзакцию с корнем дерева. Это особенно важно, так как с увеличением количества транзакций, логарифм по основанию 2 от количества транзакций растет значительно медленнее. Это позволяет Bitcoin-узлам эффективно производить пути из 10 или 12 хешей (320-384 байт), которые могут предоставить доказательства для более чем тысячи транзакций в блоке размером 1 Мб.

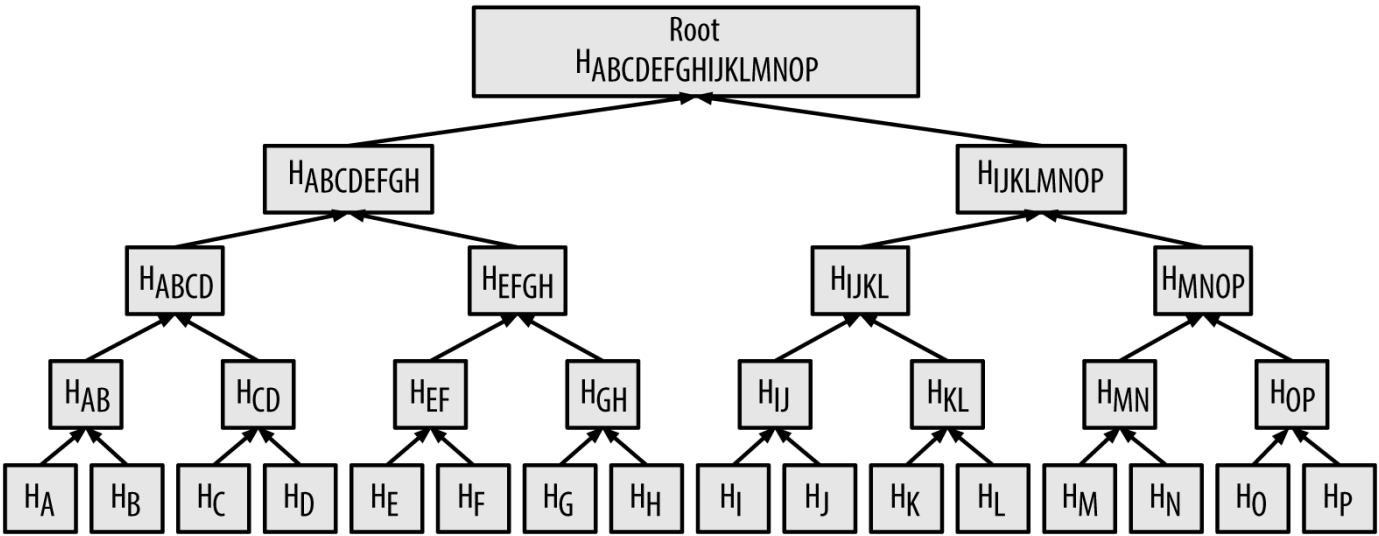


Figure 4. *Дерево Меркле, обобщающее множество элементов данных*

В [Путь Меркле, используется для доказательства включения элемента данных](#), узел может убедиться, что транзакция K входит в блок вычислив путь Меркле длиной всего лишь четыре 32-байтовых хеша (всего 128 байт). Путь состоит из четырех хэшей (отмечены синим цветом в [Путь Меркле, используется для доказательства включения элемента данных](#)) H_L , H_{IJ} , H_{MNOP} и $H_{ABCDEFGH}$. С помощью этих четырех хэшей, предоставленных в качестве пути аутентификации, любой узел может убедиться, что H_K (отмечен зеленым цветом на схеме) включен в корень меркле путем вычисления четырех дополнительных попарных хэшей H_{KL} , H_{IJKL} , $H_{IJKLMNOP}$ и корень Меркле (отмечен на диаграмме пунктирной линией).

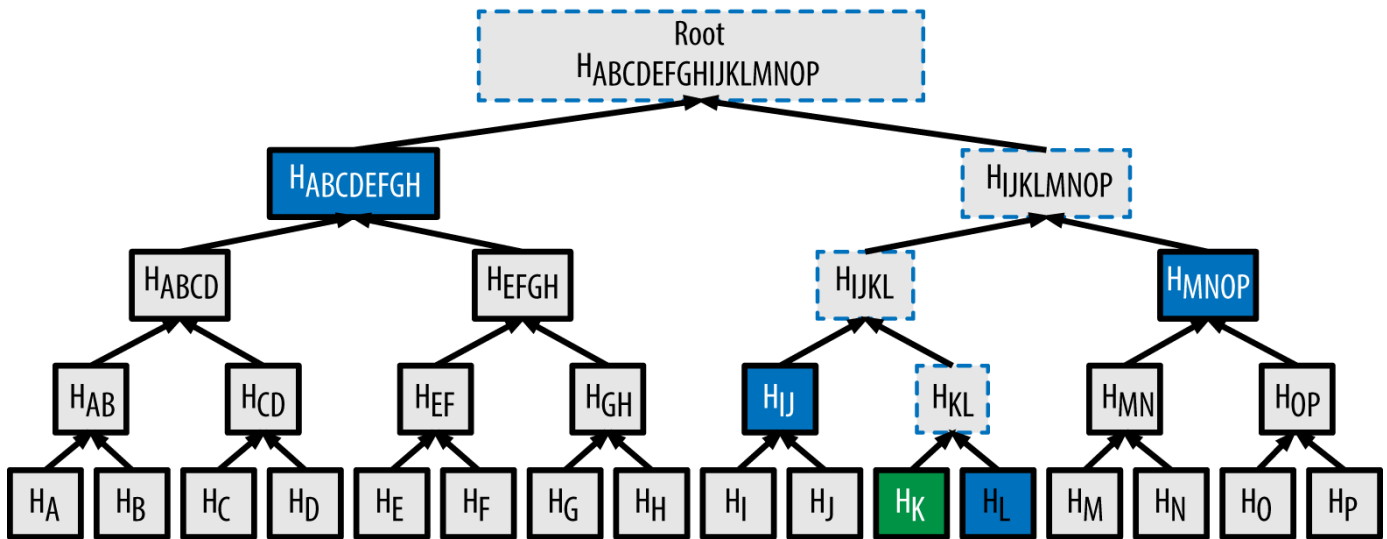


Figure 5. Путь Меркле, используется для доказательства включения элемента данных

Код из [Построение дерева Меркле](#) демонстрирует процесс создания дерева Меркле из хешей листовых вершин вплоть до корня, используя некоторые вспомогательные функции из библиотеки libbitcoin.

Example 1. Построение дерева Меркле

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

bc::hash_digest create_merkle(bc::hash_list& merkle)
{
    // Stop if hash list is empty.
    if (merkle.empty())
        return bc::null_hash;
    else if (merkle.size() == 1)
        return merkle[0];

    // While there is more than 1 hash in the list, keep looping...
    while (merkle.size() > 1)
    {
        // If number of hashes is odd, duplicate last hash in the list.
        if (merkle.size() % 2 != 0)
            merkle.push_back(merkle.back());
        // List size is now even.
        assert(merkle.size() % 2 == 0);

        // New hash list.
        bc::hash_list new_merkle;
        // Loop through hashes 2 at a time.
        for (auto it = merkle.begin(); it != merkle.end(); it += 2)
        {
            // Join both current hashes together (concatenate).
```

```

        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());
        // Hash both of the hashes.
        bc::hash_digest new_root = bc::bitcoin_hash(concat_data);
        // Add this to the new list.
        new_merkle.push_back(new_root);
    }
    // This is the new list.
    merkle = new_merkle;

    // DEBUG output -----
    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;
    // -----
}
// Finally we end up with a single item.
return merkle[0];
}

int main()
{
    // Replace these hashes with ones from a block to reproduce the same merkle root.
    bc::hash_list tx_hashes{
        bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000000"),
        bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000011"),
        bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000022"),
    };
    const bc::hash_digest merkle_root = create_merkle(tx_hashes);
    std::cout << "Result: " << bc::encode_hex(merkle_root) << std::endl;
    return 0;
}

```

В [Компиляция и запуск кода примера](#) показан результат компиляции и запуска merkle.cpp.

Example 2. Ромпиляция и запуск кода примера

```
$ # Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Запуск на выполнение программы merkle
$ ./merkle
Текущий список хэша Меркле:
32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Текущий список хэша Меркле:
d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

Эффективность деревьев Меркле становится очевидной с увеличением масштаба. В [Эффективность дерева Меркле](#) показано количество данных, которые должны быть заменены на пути Меркле, для доказательства присутствия транзакции в блоке.

Table 3. Эффективность дерева Меркле

Количество транзакций	Прибл. размер блока	Размер пути (в хэшах)	Размер пути (в байтах)
16 транзакций	4 килобайта	4 хэшей	128 байт
512 транзакций	128 килобайт	9 хэшей	288 байт
2048 транзакций	512 килобайт	11 хэшей	352 байт
65535 транзакций	16 мегабайт	16 хэшей	512 байт

Как можно видеть из таблицы, в то время как размер блока быстро увеличивается, с 4 КБ и 16 транзакций до 16 МБ, вмещающих 65535 сделок, путь Меркле, необходимый для удостоверения включения транзакции в блок растет гораздо медленнее, со 128 байт до лишь 512 байт. Используя деревья Меркле, узел может загрузить только заголовки блоков (80 байт на блок) и по-прежнему быть в состоянии идентифицировать включение транзакции в блок путем получения короткого пути Меркле с полного узла, без передачи большей части blockchain. Узлы, которые не поддерживают полный blockchain, называются SPV-узлами и используют пути Меркле проверки транзакций без загрузки полных блоков.

Деревья Меркле и Упрощенная Проверка Оплаты (SPV)

Деревья Меркле широко используются SPV-узлами. SPV-узлы не содержат все транзакции и не

загружают полные блоки, только заголовки. Для того, чтобы убедиться, что транзакция включена в блок, эти узлы используют пути Меркле без необходимости загрузки всех транзакций блока.

Рассмотрим, например, SPV-узел, который заинтересован во входящих платежах на адрес, содержащийся в его кошельке. SPV-узел установит фильтр Блума на соединения с другими пирами для того, чтобы ограничить транзакции только теми, которые содержат адреса, представляющие интерес. Когда узел видит транзакцию, которая соответствует фильтру Блума, он пошлет этот блок, используя сообщение merkleblock. Сообщение merkleblock содержит заголовок блока, а также путь Меркле, который связывает интересующую транзакцию с корнем Меркле в блоке. Узел SPV может использовать этот путь Меркле для связывания транзакции и блока и проверки вхождения транзакции в блок. SPV-узел также использует заголовок блока, чтобы связать блок к остальной частью blockchain. Сочетание этих двух звеньев, между транзакцией и блоком, а также между блоком и blockchain, доказывает, что транзакция записана в blockchain. В общем, SPV-узел получил меньше килобайта данных заголовка блока и пути Меркле, т.е. чем в тысячу раз меньше, чем потребовалось бы для передачи полного блока (в данный момент это около 1 мегабайта).

Майнинг и Консенсус

Введение

Майнинг — это процесс, в результате которого эмитируются новые биткойны. Майнинг также служит для защиты системы Bitcoin от мошеннических транзакций или транзакций, тратящих одни и те же биткойны дважды. Майнеры обеспечивают вычислительную мощность сети Bitcoin в обмен на возможность получить вознаграждение.

Майнеры валидируют новые транзакции и записывают их в общую бухгалтерскую книгу. Новый блок, содержащий транзакции, которые произошли с момента последнего блока, возникает в среднем каждые 10 минут, таким образом, добавляя эти сделки в blockchain. Транзакции, которые становятся частью блока и добавленные в blockchain считаются "подтвержденными", что позволяет новым владельцам биткойнов распоряжаться этими биткойнами уже с помощью собственных транзакций.

Майнеры получают два вида вознаграждения: новые монеты, создаваемые с каждым новым блоком и комиссионные со всех операций, входящих в блок. Для того, чтобы заработать это вознаграждение, майнеры наперегонки решают сложную математическую задачу, основанную на криптографическом хеш-алгоритме. Решение этой задачи, называемое доказательством работы (proof of work), встраивается в новый блок и выступает в качестве доказательства того, что майнер потратил значительное количество вычислений. Соревнование по поиску решения на задачу, а также право на запись операций в blockchain являются основой для модели безопасности Bitcoin.

Процесс создания новых монет напоминает добычу полезных ископаемых, которые извлекать из земли с течением времени становится все труднее. Именно из-за этой аналогии майнинг (mining ['maɪnɪŋ] — разработка месторождения, англ. прим пер.) и получил свое название. Новые биткойны создаются в процессе майнинга подобно тому, как центральный банк создает новые деньги путем печатания банкнот. Количество биткойнов, которые создаются вместе с новым блоком уменьшается примерно через каждые четыре года (или точнее, каждые 210000 блока). Сначала с блоком появлялось 50 биткойнов, а в ноябре 2012 года это количество уменьшилось в два раза до 25 Bitcoin за блок и в следующий раз уменьшится до 12,5 Bitcoin за блок-то в 2016 году. Согласно этой формулы, награда за майнинг будет экспоненциально убывать до примерно 2140 года, когда будут достигнут предел (2099999998 млн монет). После 2140 новые монеты создаваться не будут.

Майнеры также забирают себе комиссионные за транзакции. Каждая транзакция может включать в себя комиссионные, в виде разницы между входами и выходами транзакции. Майнер, нашедший блок, забирает себе комиссионные за транзакции, которые он включает в блок. На сегодняшний день комиссионные составляют 0.5% или меньше от дохода майнеров, подавляющее же большинство доходов получается от награды за блок. Однако, награда за блок с течением времени уменьшается, а количество транзакций в блок будет только увеличивается так, что со временем все большая часть доходов майнеров будет поступать от

комиссионных. В 2140-ом году, вся прибыль от майнинга будет в виде комиссионных за транзакции.

Хотя майнинг и стимулируется наградой, основной целью майнинга является не награждение майнеров или создание новых монет. Майнинг является основным процессом децентрализованной информационной расчетной палаты, с помощью которого валидируются транзакции. Майнинг защищает систему Bitcoin и позволяет появление консенсуса в масштабах всей сети без центральной власти.

Майнинг—это изобретение, которое делает Bitcoin особенным. Его децентрализованный механизм безопасности является основой пиринговых цифровых наличных. Награда в виде новых монет и комиссионных за транзакции является тем стимулом, который обуславливает действия майнеров и обеспечивает безопасность сети, при одновременном создании денежного предложения.

В этой главе мы сначала рассмотрим майнинг в качестве механизма денежной эмиссии, а затем рассмотрим наиболее важную функцию майнинга: механизм децентрализованного консенсуса, который лежит в основе безопасности Bitcoin.

Экономика Биткоин и создание валюты

Биткоины "чеканятся" по мере создания каждого нового блока на фиксированной медленно убывающей скорости. Каждый блок, генерируется в среднем каждые 10 минут и содержит совершенно новые биткоины, созданные из ничего. Каждые 210,000 блоков, или примерно раз в четыре года, эмиссия уменьшается на 50%. В первые четыре года работы сети, каждый блок содержал 50 новых биткоинов.

В ноябре 2012 года, скорость эмиссии была снижена до 25 биткоинов в блоке, и она уменьшится снова до 12,5 биткоинов в блоке номер 420,000, который будет найден в 2016 году. Темпы эмиссии будут уменьшаются экспоненциально 64 "уполовинивания" вплоть до блока номер 13,230,000 (который будет найден примерно в 2137 году), когда он достигает минимального денежной единицы 1 Satoshi. Наконец, через 13.44 млн блоков, в примерно 2140, в системе уже будет обращаться почти 2,099,999,997,690,000 сатоши, или почти 21 млн биткоинов. После этого блоки не будут содержать новые биткоины, и майнеры будут получать вознаграждения исключительно посредством комиссий. В [Темпы эмиссии биткоинов со временем уменьшаются геометрически](#) показател общее количество биткоинов в обращении в течение времени.

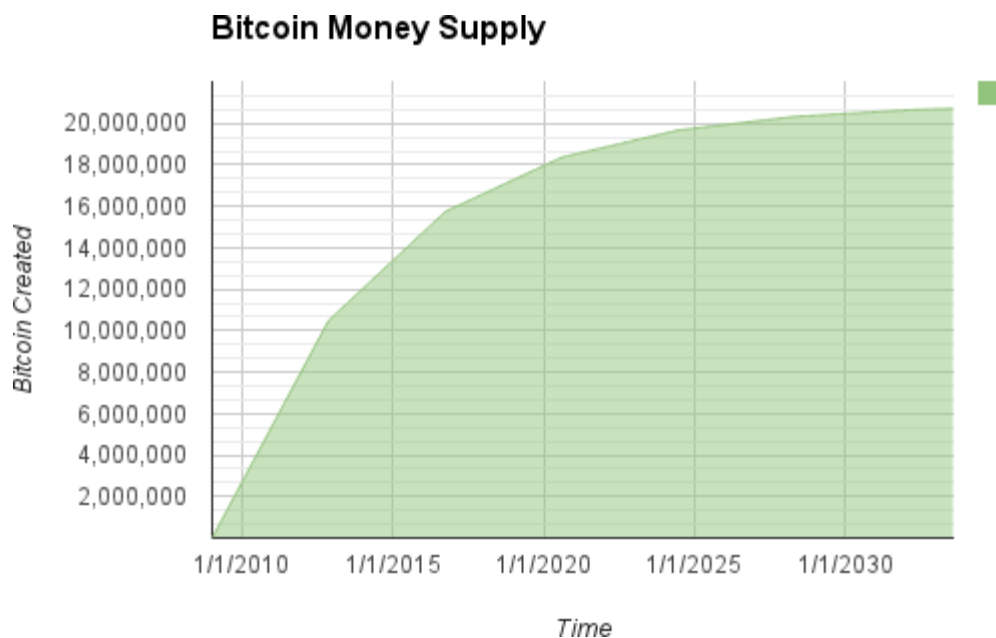


Figure 1. Темпы эмиссии биткоинов со временем уменьшаются геометрически

NOTE

Максимальное количество добываемых монет выступает *верхним пределом* возможной награды. На практике, майнер может найти блок и намеренно взять меньше, полного вознаграждения. Подобные случаи в истории уже были и могут случиться в будущем, что приведет к снижению общей эмиссии.

В примере кода в [Сценарий для расчета общего количества биткоинов](#) мы рассчитываем общее количество биткоинов, которые когда-либо появятся.

Example 1. Сценарий для расчета общего количества биткоинов

```
# Original block reward for miners was 50 BTC
start_block_reward = 50
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000

def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
    while current_reward > 0:
        total += reward_interval * current_reward
        current_reward /= 2
    return total

print "Total BTC to ever be created:", max_money(), "Satoshis"
```

В [Запускаем скрипт max_money.py](#) показан вывод скрипта.

```
$ python max_money.py  
Total BTC to ever be created: 2099999997690000 Satoshis
```

Конечная и убывающая эмиссия создает фиксированное денежное предложение, что препятствует инфляции. В отличие от фиатных валют, которые могут быть напечатаны в любых количествах центральным банком, эмиссия Bitcoin происходит алгоритмически.

Дефляционные деньги

Самое важное и обсуждаемое следствие фиксированной и уменьшающейся денежной эмиссии, является то, что валюта будет иметь тенденцию быть по своей сути *дефляционной*. Дефляция — это феномен роста стоимости из-за несоответствия спроса и предложения, что взвинчивает стоимость (и обменный курс) той или иной валюты. Противоположность инфляции, дефляция цен означает, что покупательная способность денег растет со временем.

Многие экономисты утверждают, что дефляционная экономика — это бедствие, которого следует избегать любой ценой. Это происходит потому, что в период быстрой дефляции, люди склонны копить деньги вместо того, чтобы тратить, в надежде, что цены упадут. Подобное явление наблюдалось в Японии во время "потерянного десятилетия", когда полный крах спроса столкнул национальную валюту в дефляционную спираль.

Эксперты Биткоин утверждают, что дефляция не плоха сама по себе. Для фиатных валют существует возможность неограниченной эмиссии, поэтому войти в дефляционную спираль очень трудно, только при условии полного обвала спроса и нежелания печатать деньги. Дефляция в Биткоин не вызвана падением спроса, но происходит вследствие предсказуемо ограниченного предложения.

На практике стало очевидным, что инстинкт накопительства, вызванный дефляционной природой валюты может быть преодолен скидками от продавцов до тех пор, пока скидка не преодолевает инстинкт накопительства покупателя. Поскольку продавец также мотивирован копить, скидка становится равновесной ценой, при которой уравниваются инстинкты накопительства обеих сторон. Со скидкой 30% в цене, большинство розничных Биткоин-торговцев не испытывают трудности с преодолением инстинкта накопительства и способности приносить доход. Предстоит выяснить, является ли дефляционной аспект валюты действительной проблемой, когда он не обусловлен быстрым сокращением экономики.

Децентрализованный консенсус

В предыдущей главе мы познакомились с устройством блокчейн, глобальной публичной бухгалтерской книгой (списком) всех транзакций, которые все в сети Биткоин принимают как авторитетный источник информации о собственности.

Но как все участники сети договариваются об единой универсальной «правде» о том, кто чем владеет, без необходимости доверия кому-либо? Все традиционные платежные системы зависят от модели доверия, в которой имеется центральный орган, обеспечивающий клиринговые функции, верифицирующий и засчитывающий все транзакции. В Биткоин нет центральной власти, однако каждый полный узел содержит полную копию публичной бухгалтерской книги, которой он может доверять. Blockchain не создается центральным органом, но собран независимо каждым узлом в сети. Так или иначе, каждый узел сети, действуя на основе информации, передаваемой через небезопасные сетевые соединения, может прийти к одинаковому выводу и собрать копию той же публичной бухгалтерской книги, что и все остальные. В этой главе рассматривается процесс, с помощью которого сеть Bitcoin достигает глобального консенсуса без необходимости существования центральной власти.

Главным изобретением Сатоши Накамото является децентрализованный механизм *неявного консенсуса*. Неявный, поскольку нет никаких выборов или фиксированного момента возникновения консенсуса. Вместо этого консенсус является неявным артефактом асинхронного взаимодействия тысяч независимых узлов, следующих простым правилам. Все свойства Bitcoin, в том числе валюты, транзакций, платежей и модели безопасности, независимой от центральной власти, вытекают из данного изобретения.

Децентрализованный консенсус Bitcoin возникает из взаимодействия четырех процессов, которые происходят независимо друг от друга на узлах сети:

- Независимая проверка каждой транзакции, каждым полным узлом, на основе обширного перечня критериев
- Независимое объединение этих транзакций в новые блоки майнерами, в сочетании с демонстрацией вычислений с помощью алгоритма доказательства работы
- Независимая проверка новых блоков на каждом узле и сборка в цепочку
- Независимый выбор, каждым узлом, цепочки с наибольшим затраченным количеством вычислительной работы

В следующих разделах мы рассмотрим эти процессы и как они взаимодействуют для создания общесетевого консенсуса, который позволяет любому Bitcoin-узлу поддерживать свою собственную копию авторитетной, доверенной, публичной, глобальной бухгалтерской книги.

Независимая проверка сделок

В [\[transactions\]](#), мы могли видеть, как ПО кошелька создает транзакции путем отбора UTXO,

создавая соответствующие отпирательные сценарии, а затем строит новые выходы. Полученная транзакция затем отправляется на соседние узлы и далее распространяется по всей сети Bitcoin.

Тем не менее, перед пересылкой транзакции своим пирам, каждый Bitcoin узел сначала ее проверит. Это гарантирует, что по сети передаются только валидные транзакции, в то время как недопустимые транзакции отбрасываются на первом же узле, который сталкивается с подобными.

Каждый узел проверяет каждую транзакцию по списку критериев:

- Синтаксис транзакции и структура данных должны быть корректными.
- Списки входов и списки выходов не должны быть пустыми.
- Размер транзакции в байтах должен быть меньше MAX_BLOCK_SIZE.
- Значение каждого выхода, а также общее, должно быть в пределах допустимого диапазона значений (менее 21 млн монет, более чем 0).
- Ни один из входов не должен иметь хэш=0, N=-1 (coinbase-транзакции не должны быть ретранслированы).
- nLockTime меньше или равен INT_MAX.
- Размер транзакции в байтах должен быть больше или равен 100.
- Количество подписей, содержащихся в транзакциях сделки меньше лимита.
- Отпирательный сценарий (scriptSig) может только помещать цифры в стек, а запирающий сценарий (scriptPubkey) должен соответствовать isStandard формам (это отвергает "нестандартные" транзакции).
- Соответствующая транзакция должна существовать в пуле или в блоке основной ветви.
- Для каждого входа, если упоминаемый выход существует в любой другой транзакции в пуле, транзакция должна быть отклонена.
- Для каждого входа, обратиться к основной ветви и пулу транзакций в поисках указанной выходной транзакции. Если выходная транзакция отсутствует для какого-либо входа, она объявляется бесхозной и добавляется в сиротский пул, если ее там еще нет.
- Для каждого входа, если упоминаемый выход транзакции является coinbase-выходом, он должен иметь не менее COINBASE_MATURITY (100) подтверждений.
- Для каждого входа должен существовать обозначенный выход и он не должен быть потрачен до этого.
- Используя перечисленные выходные транзакции для получения входных значений, проверить, что каждое значение входа, а также сумма, находятся в допустимом диапазоне значений (менее 21 миллиона монет, больше чем 0).
- Отклонить, если сумма входных значений меньше суммы выходных значений.
- Отклонить, если комиссия за транзакцию будет слишком маленькой, чтобы попасть в

пустой блок.

- Отпирательный сценарий для каждого входа должен валидироваться относительно соответствующего выходного запирающего сценария

Детали этих условий можно найти в функциях `AcceptToMemoryPool`, `CheckTransaction`, и `CheckInputs` базового клиента. Обратите внимание, что условия изменяются с течением времени для противостояния новым видам атак на отказ в обслуживании или для разрешения новых видов транзакций.

Независимо проверяя каждую транзакцию по мере поступления и до ее распространения дальше, каждый узел строит пул действительных (но неподтвержденных) транзакций, известный как *transaction pool*, *memory pool* или *mempool*.

Майнинговые узлы

Некоторые из узлов сети Bitcoin являются специализированными и называются *майнерами*. В [\[ch01_intro_what_is_bitcoin\]](#) мы познакомились с Дзином, студентом компьютерного факультета из Шанхая, который занимается майнингом Bitcoin. Цзинь зарабатывает Bitcoin при помощи специализированного оборудования. Его майнинговое аппаратное обеспечение подключено к серверу, на котором запущен полный Bitcoin-узел. Майнинг также возможен и без наличия полного узла, как мы увидим в [Майнинговые пулы](#). Как и любой другой полный узел, узел Цзиня принимает и распространяет неподтвержденные транзакции в сети Bitcoin. Однако узел Цзиня, также способен объединять эти транзакции в новые блоки.

Узел Цзиня прослушивает сеть Bitcoin на предмет наличия новых блоков так же, как это делают все остальные узлы. Тем не менее, приход нового блока имеет особое значение для майнингового узла. Конкуренция среди майнеров заканчивается как только приходит новый блок, который выступает в качестве объявления победителя. Для майнеров, получить новый блок означает, что кто-то другой выиграл соревнование, а они проиграли. Тем не менее, конец одного цикла соревнований означает также начало следующего раунда.

Агрегирование транзакций в блоки

После проверки транзакций, узел Bitcoin добавит их к *пулу памяти* или *пулу транзакций*, где транзакции ожидают, пока они не смогут быть включены в блок. Узел Цзиня собирает, проверяет, и передает новые транзакции так же, как любой другой узел. В отличие от других узлов, однако, узел Цзиня будет агрегировать эти транзакции в *блок-кандидат*.

Давайте проследим блоки, которые были созданы, когда Алисой купила чашечку кофе в кафе Боба (см. [\[cup_of_coffee\]](#)). Транзакция Алисы была включена в блок 277316. В целях демонстрации давайте предположим, что этот блок был найден на майнинговом оборудовании Цзиня и проследим как транзакция Алисы стала частью этого нового блока.

Майнинговый узел Цзиня поддерживает локальную копию blockchain, список всех блоков, созданных с момента запуска системы Bitcoin в 2009 и до сего момента. В момент, когда Алиса

покупает чашку кофе, узел Цзиня собрал цепь до блока 277314. Узел Цзиня прослушивает транзакции, пытаясь добыть новый блок, а также прослушивает блоки, обнаруженные другими узлами. Получение блока 277315 означает конец соревнования за этот блок и начало конкурса по поиску блока 277316.

В течение предыдущих 10 минут, пока узел Цзиня искал решение для блока 277315, он также собирал транзакции в рамках подготовки к следующему блоку. К настоящему моменту он собрал несколько сотен транзакций в пуле памяти. После получения блока 277315 и его проверки, узел Цзиня также проверит все транзакции в пуле и удалит все попавшие в блок 277,315. Любые транзакции, оставшиеся в пуле являются неподтвержденными и ждут быть записанными в новый блок.

Узел Цзиня немедленно создает новый пустой блок, кандидат на блок номер 277,316. Этот блок называется блоком-кандидатом, так как это еще не валидный блок, поскольку он не содержит действительное доказательство работы. Блок объявляется валидным только тогда, когда шахтер майнеру удастся найти решение для доказательства работы.

Возраст транзакции, комиссионные и приоритет

Для создания блока-кандидата, Биткоин-узел Цзиня выбирает транзакции из пула памяти согласно приоритету. Транзакции приоритизируются на основе «возраста» UTXO, который тратится на их входах, что позволяет старым и стоимостным входам получить повышенный приоритет по сравнению с новыми и менее "ценными" входами. Приоритетные транзакции могут быть отправлены вообще без комиссионных в случае, если в блоке достаточно места.

Приоритет транзакции рассчитывается как сумма стоимости и возраста входов, разделенная на общий размер транзакции:

$$\text{Приоритет} = \text{Сумма(стоимость на входе * возраст входа)} / \text{размер транзакции}$$

В этом уравнении значение входа измеряется с помощью базовой единицы satoshis (одна стомиллионная часть Bitcoin). Возраст UTXO — это число блоков, прошедших с момента, когда UTXO был записан в blockchain, отражая "глубину" в цепочке блоков. Размер транзакции измеряется в байтах.

Для того, чтобы транзакция считалась "высокоприоритетной", ее приоритет должен быть больше 57,600,000, что соответствует одному биткоину (100 млн сатоши), возрастом один день (144 блоков), с общим размером транзакции 250 байт:

$$\text{Высокий приоритет} > 100,000,000 \text{ сатоши} * 144 \text{ блока} / 250 \text{ байтов} = 57,600,000$$

Первые 50 килобайт пространства транзакций в блоке отведены для высокоприоритетных операций. Узел Цзинь отведет первые 50 килобайт под наиболее приоритетные транзакции, независимо от комиссионных. Это позволяет высокоприоритетным транзакциям быть

обработанным, даже если они не принесут комиссионных.

Майнинговый узел Цзин затем заполняет остальную часть блока до максимального размера блока (MAX_BLOCK_SIZE) транзакциями, которые содержат, по крайней мере, минимальные комиссионные, отдавая приоритет с самой высокой суммой за килобайт.

Если в блоке останется свободное место, узел Цзин может решить заполнить его транзакциями без комиссии. Некоторые майнеры могут решить вставить такие транзакции в блок, а другие могут решить проигнорировать.

Любые транзакции, оставшиеся в пуле памяти, после заполнения блока, останутся в пуле до включения в следующий блок. Поскольку транзакции остаются в пуле памяти, их входы "стареют", так как UTXO, которые эти входы тратят, "погружаются" глубже в blockchain по мере добавления новых блоков к вершине. Поскольку приоритет транзакций зависит от возраста его входов, транзакции, оставшиеся в пуле будут стареть и, следовательно, их приоритет расти. В конце концов транзакция без комиссионных может достичь достаточно высокого приоритета и попасть в блок бесплатно.

Bitcoin-транзакции не содержат тайм-аута истечения. Транзакция, действующая в настоящее время будет действовать бессрочно. Тем не менее, если транзакция распространилась по сети только один раз, он будет сохраняться до тех пор, пока удерживается в пуле памяти майнерского узла. Если майнерский узел будет перезапущен, его пул памяти очистится. Хотя валидная транзакция могла быть распространена по сети, если она не была исполнена, она может в конце концов не оказаться в пулах никакого из майнеров. Ожидается, что программное обеспечение кошельков, должно ретранслировать подобные транзакции или реконструировать их с более высокими комиссионными в случае, если они не окажутся успешно исполнены в течение разумного периода времени.

Когда узел Цзиня агрегирует все транзакции из пула памяти, новый блок-кандидат получит 418 транзакций с общей суммой комиссионных равной 0.09094928 Bitcoin. Вы можете увидеть этот блок в blockchain с помощью интерфейса командной строки клиента Bitcoin Core, как это показано в [Блок 277,316](#).

```
$ bitcoin-cli getblockhash 277316
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

[illegible]

Порождающая транзакция

Первая транзакция, добавляемая в блок, является специальной и называется *порождающей транзакцией* или *coinbase-транзакцией*. Эта транзакция создается узлом Цзин и становится его наградой за майнинг. Узел Цзин создает порождающую транзакцию с платежом на свой собственный кошелек: "Заплатить Цзиню сумму 25.09094928 биткоинов". Общая сумма вознаграждения, которую Цзин получит за блок представляет собой сумму coinbase-вознаграждения (25 новых биткоинов) и комиссионные (0.09094928) со всех транзакций, входящих в блок, как показано на [Порождающая транзакция](#):

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```


Награда Coinbase и комиссионные

Для создания порождающей транзакции, узел Цзиня сначала вычисляет общую сумму комиссионных, сложив все входы и выходы 418-ти сделок, добавленных к блоку. Комиссионные рассчитываются как:

Общие комиссионные = Сумма(входы) - Сумма(выходы)

В блоке 277,316, общие комиссионные составляют 0.09094928 биткоинов.

Затем узел Цзиня вычисляет правильную награду для нового блока. Вознаграждение рассчитывается на основе высоты блока, начиная с 50 Bitcoins на блок и уменьшается наполовину каждые 210000 блоков. Поскольку данный блок находится на высоте 277316, вознаграждение равняется 25 Bitcoins.

Вычисления можно увидеть в функции `GetBlockSubsidy` в клиенте Bitcoin Core, как это показано в [Вычисление награды за блок - функция GetBlockSubsidy, Bitcoin Core Client, main.cpp](#).

Example 5. Вычисление награды за блок - функция GetBlockSubsidy, Bitcoin Core Client, main.cpp

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Вернуть сумму комиссионных без награды за блок
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // Субсидия уменьшается в два раза каждые 210000 блоков (примерно
    // каждые 4 года).
    nSubsidy >>= halvings;
    return nSubsidy;
}
```

Первоначальная субсидия рассчитывается в сатошах путем умножения 50 на константу COIN (=100 млн сатоши). Это устанавливает начальное вознаграждение (`nSubsidy`) в 5 миллиардов сатоши.

Далее, функция вычисляет количество прошлых уполовиниваний, путем деления текущей высоты блока на уполовинивающий интервал (`SubsidyHalvingInterval`). В случае с блоком 277316 и уполовинивающим интервалом 210000 блоков, результат равен 1-му уполовиниванию.

Разрешено максимально 64 уполовинивания, так что код устанавливает нулевое

вознаграждение (возвращает только комиссионные), если число уполовиниваний превышает 64.

Далее, функция использует оператор побитового сдвига вправо для того, чтобы разделить вознаграждение (nSubsidy) на два для каждого раунда уполовинивания. В случае с блоком 277316, побитовый сдвиг вправо награды 5 млрд сатоши на один бит даст результат 2.5 млрд сатоши или 25 биткоинов.

И, наконец, награда coinbase (nSubsidy) складывается с комиссионными за транзакции, а сумма возвращается.

Структура порождающей транзакции

Используя эти вычисления, узел Цзинга затем создает порождающую транзакцию самому себе на сумму 25.09094928 биткоинов.

Как вы можете видеть из [Порождающая транзакция](#), порождающая транзакция имеет специальный формат. Вместо входа транзакции с указанием предыдущего UTXO для траты, у него есть "coinbase" вход. Мы изучили входы транзакций в [\[tx_in_structure\]](#). Давайте сравним вход обычной транзакции со входом порождающей транзакции. В [Структура "нормального" вход транзакции](#) показана структура обычной транзакции, а в [Структура входа порождающей транзакции](#) структура входа порождающей транзакции.

Table 1. Структура "нормального" вход транзакции

Размер	Поле	Описание
32 байта	Хэш транзакции	Указатель на транзакцию, содержащую UTXO, который будет потрачен
4 байта	Номер выхода	Номер UTXO для траты, начиная с 0
1-9 байтов (VarInt)	Размер отпирающего скрипта	Длина скрипта в байтах
Переменная	Отпирающий сценарий	Сценарий, который выполняет условия запирающего UTXO сценария.
4 байта	Номер последовательности	В настоящее время отключенная функция замены транзакции, установлен в 0xFFFFFFFF

Table 2. Структура входа порождающей транзакции

Размер	Поле	Описание
32 байта	Хэш транзакции	Все биты равны нулю: не ссылка на хэш транзакции
4 байта	Номер выхода	Все биты являются выставлены в единицу: 0xFFFFFFFF
1-9 байт (VarInt)	Размер данных Coinbase	Длина данных coinbase, от 2 до 100 байт
Переменная	Данные Coinbase	Произвольные данные, используемые для дополнительного одноразового числа и тегов майнеров в v2 блоки должны начинаться с высоты блока
4 байта	Номер в последовательности	Установлен в 0xFFFFFFFF

В порождающей транзакции, первые два поля установлены в значения, которые не представляют собой ссылку на UTXO. Вместо "хеша транзакции" первое поле заполняется 32 байтами установленными в нуль. Поле "Output Index" заполняется 4 байтами со значением 0xFF (255 в десятичной системе счисления). "Отпирающий Сценарий" заменяется данными coinbase, произвольным полем данных, используемым майнерами.

Данные Coinbase

Порождающие транзакции не содержат поля опирающего сценария (так называемого, scriptSig). Вместо этого это поле заменяется данными coinbase, которые должны быть длиной от 2 до 100 байт. За исключением первых нескольких байтов, остальная часть данных coinbase может быть использована майнерами любым способом; это произвольные данные.

В блоке генезиса, например, Сатоши Накамото поместил в данные coinbase следующий текст "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks", используя его в качестве доказательства даты и чтобы передать некое сообщение. В настоящее время майнеры используют данные coinbase для дополнительных значений nonce и строк, идентифицирующих майнинговый пул, как мы увидим в следующих разделах.

Первые несколько байтов coinbase использовались произвольным способом, но это уже не так. Согласно Bitcoin Improvement Proposal 34 (BIP0034), блоки второй версии (блоки с полем версии установленным в 2) должны содержать индекс высоты блока в качестве операции "push" сценариев в начале поля coinbase.

В блоке 277316 мы видим, что coinbase (см. [Порождающая транзакция](#)), который находится в поле "Отпирающий Сценарий" или scriptSig входа транзакции, содержит шестнадцатеричное значение 03443b0403858402062f503253482f. Давайте декодируем это значение.

Первый байт, 03, дает команду движку сценариев поместить следующие три байта в стек сценария (см. [\[tx_script_ops_table_pushdata\]](#)). Следующие три байта, 0x443b04, представляют собой высоту блока закодированную в формате little-endian (сначала младший байт). В обратном порядке байтов результат выглядит 0x043b44, что в десятичной системе счисления равняется 277316.

Следующие несколько шестнадцатеричных цифр (03858402062) используются для кодирования дополнительного *nonce* (см. [Решение Extra Nonce](#)), или случайного значения, используемого для подходящего решения задачи доказательства работы.

Заключительная часть данных coinbase (2f503253482f) — это строка в кодировке ASCII /P2SH/, которая указывает на то, что майнинг-узел, который нашел этот блок поддерживает *pay-to-script-hash* (P2SH), улучшение описываемое BIP0016. Внедрение совместимости с P2SH потребовало "голосования" среди майнеров для одобрения либо BIP0016 либо BIP0017. Те, кто одобрял реализацию BIP0016 должны были включать /P2SH/ в свои данные coinbase. Те же, кто одобрял реализацию P2SH согласно BIP0017 должны были включать в coinbase строку p2sh/CHV. В конечном счете в качестве победителя был выбран BIP0016, но многие майнеры продолжили включать строку /P2SH/ в coinbase, казывая на поддержку этой функции.

В [Выделение coinbase-данных из блока генезиса](#) используется библиотека `libbitcoin`, описанная в [\[alt_libraries\]](#) для извлечения coinbase данных из блока генезиса и показывающая сообщение от Сатоши Накамото. Обратите внимание, что `libbitcoin` содержит статическую копию блока генезиса, так что пример кода может извлечь блок генезиса непосредственно из библиотеки.

Example 6. Выделение coinbase-данных из блока генозиса

```
/*
    Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::block_type block = bc::genesis_block();
    // Genesis block contains a single coinbase transaction.
    assert(block.transactions.size() == 1);
    // Get first transaction in block (coinbase).
    const bc::transaction_type& coinbase_tx = block.transactions[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs.size() == 1);
    const bc::transaction_input_type& coinbase_input = coinbase_tx.inputs[0];
    // Convert the input script to its raw format.
    const bc::data_chunk& raw_message = save_script(coinbase_input.script);
    // Convert this to an std::string.
    std::string message;
    message.resize(raw_message.size());
    std::copy(raw_message.begin(), raw_message.end(), message.begin());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}
```

Скомпилируем код компилятором GNU C++ и запустим получившийся исполняемый файл, как показано на [Сборка и запуск примера satoshi-words](#).

Example 7. Сборка и запуск примера satoshi-words

```
$ # Скомпилируем код
$ g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Запустим исполняемый файл
$ ./satoshi-words
^D    <GS>^A^DThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Создание заголовка блока

Для того, чтобы собрать заголовок блока, майнинговый узел должен заполнить шесть полей, как указано в [Структура заголовка блока](#).

Table 3. Структура заголовка блока

Размер	Поле	Описание
4 байта	Версия	Номер версии для отслеживания обновлений программного обеспечения/протоколов
32 байта	Хэш Предыдущего Блока	Ссылка на хэш предыдущего (родительского) блока в цепочке
32 байта	Корень дерева Меркле	Хэш корня дерева Меркле транзакций этого блока
4 байта	Временная метка	Приблизительное время создания этого блока (секунды от эпохи Unix)
4 байта	Целевая сложность	Целевая сложность алгоритма доказательства работы для этого блока
4 байта	Некое случайное число (nonce)	Счетчик используется в алгоритме доказательства работы

Как только блок номер 277316 найден, в него прописывается номер версии блока 2. В структуре данных это поле занимает 4 байта и записывается в фомате little-endian, т.е. 0x02000000.

Далее, майнинговый узел должен добавить "Хеш Предыдущего Блока". То есть хеш заголовка блока номер 277315, предыдущего блока, полученного из сети, который узел Цзинга принял и выбрал в качестве родителя для кандидата блока номер 277,316. Хеш заголовка блока для блока 277315 выглядит так:

```
00000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

Следующим шагом будет обобщение всех транзакций в дереве Меркле, для того, чтобы в заголовок блока было бы возможно добавить корень Меркле. Порождающая транзакция указывается в качестве первой транзакции в блоке. После нее в блок добавляется еще 418 транзакций, в общей сложности 419. Как мы видели в [\[merkle_trees\]](#), в дереве должно быть четное количество "листьев", поэтому последняя транзакция дублируется, создавая 420 узлов, в

каждом из которых содержится хэш одной транзакции. Хэши транзакций затем объединяются в пары, создавая каждый уровень дерева, пока все транзакции не объединяются в один узел в "корне" дерева. Корень дерева Меркле обобщает все транзакции в одно значение длиной 32 байт, которое вы можете видеть в [Блок 277,316](#) под названием "Корень Меркле" и здесь:

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

Майнинговый узел затем должен добавить четырехбайтовую временную метку, в формате "Эпохи Unix", т.е. количество секунд, прошедших с 1 января 1970 года. Метка 1388185914 означает пятницу, 27 декабря 2013, 23:11:54 UTC/GMT.

Далее узел заполняет поле целевой сложности, которая требуется валидации блока с помощью доказательства работы. Сложность хранится в виде битовой экспоненциальной записи, где сначала идет 1-байтовый порядок, за которым следует 3-байтовая мантисса (коэффициент). В блоке 277316, например, сложность составляет число 0x1903a30c. Первая часть 0x19 — это шестнадцатеричный порядок, а 0x03a30c, коэффициент. Понятие целевой сложности объясняется в [Целевая Сложность и Перенацеливание](#) а битовое поле объясняется в [Представление сложности](#).

Последним находится поле нонсе, которое устанавливается в нуль.

Когда все другие поля заполнены, заголовок блока готов и можно начинать майнинг. Цель состоит в том, чтобы найти такое значение нонсе, которое дает хэш заголовка блока меньше целевой сложности. Майнинговый узел должен будет перепробовать миллиарды или триллионы значений нонсе перед тем, как найдется подходящее.

Майнинг блока

Теперь, когда блок-кандидат был построен узлом Цзиня, настал момент для майнинга блока, который сделает блок валидным. В этой книге мы изучали криптографические хэш-функции, используемые в различных аспектах системы Bitcoin. Хэш-функция SHA256 используется в процессе майнинга Bitcoin.

Проще говоря, майнинг представляет собой процесс многократного хеширования заголовка блока с постоянной подменой одного параметра, до тех пор, пока в результативный хэш не станет соответствовать определенной цели. Результат выполнения хэш-функции не может быть определен заранее, и невозможно подобрать образец, производящий определенное значение хэш-функции. Эта особенность хэш-функции означает, что единственный способ получения результата хэша, соответствующего конкретной цели состоит в подборе входных данных, пока желаемый результат хеширования не появится случайным образом.

Алгоритм доказательства работы

Алгоритм хеширования принимает на входе данные произвольной длины и производит

детерминированный результат фиксированной длины, цифровой отпечаток входа. Для любого конкретного входа, полученный хэш всегда будет одинаков, и может быть легко вычислен и проверен любой другой реализацией этого же самого алгоритма хэширования. Ключевой характеристикой криптографического хэш-алгоритма является то, что практически невозможно найти два разных входа, которые бы производили один и тот же отпечаток. Как следствие, также практически невозможно выбрать вход, таким образом, чтобы получить желаемый отпечаток, кроме как при помощи перебора.

Выход SHA256, всегда имеет длину 256 бит, независимо от размера входных данных. В [Пример SHA256](#), мы будем использовать интерпретатор Python для расчета SHA256-хэша фразы "I am Satoshi Nakamoto."

Example 8. Пример SHA256

```
$ python
```

```
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

В [Пример SHA256](#) показан результат вычисления хэша строки "I am Satoshi Nakamoto": 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Это 256-битное число представляет собой *хеш* или *дайджест* фразы и зависит от каждой части фразы. Добавление одной буквы, знака препинания или другого символа, даст другой хэш.

Теперь, если мы изменим эту фразу, мы должны ожидать совершенно другие хэши. Давайте проверим это, добавляя числа к концу нашей фразы, с помощью простого Python-скрипта из [SHA256 Скрипт для генерации множества хэшей при помощи итерации nonce](#).

Example 9. SHA256 Скрипт для генерации множества хэшей при помощи итерации nonce

```
# example of iterating a nonce in a hashing algorithm's input

import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in xrange(20):

    # add the nonce to the end of the text
    input = text + str(nonce)

    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()

    # show the input and hash result
    print input, '=>', hash
```

После запуска скрипт выдаст хеши нескольких фраз, которые получились путем добавления числа в конец текста. Приращением числа, мы можем получить различные хэши, как показано на [SHA256 Вывод скрипта генерации множества хэшей путем итерирования 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 => 60f01db30c1a0d4cbbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4aaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Все строки дают совершенно разные результаты хеширования. Они выглядят совершенно случайными, но вы можете воспроизвести результаты этого примера на любом компьютере с Python и увидеть точно такой же хэш.

Число, используемое в качестве переменной в подобном сценарии называется *nonce*. Nonce используется для варьирования результата криптографической функции, в данном случае варьируя отпечаток SHA256.

[illegible]

называем этот порог *целью* и задача состоит в том, чтобы найти хэш, который численно *меньше чем цель*. По мере уменьшения целевого значения, задача поиска хэша, численное значение которого, было бы меньше цели, становится все более и более трудной.

В качестве простой аналогии, представьте себе игру, в которой игроки несколько раз бросают пару кубиков, пытаясь выбросить значение меньше заранее указанной цели. В первом раунде, цель равна 12. Если вы не выбросите две шестерки, вы выиграли. В следующем раунде цель равна 11. Для выигрыша игроки должны бросить 10 или меньше, задача проста. Скажем, через несколько раундов цель равна уже 5. Теперь, более чем в половине случаев бросание костей даст больше 5-ти и, следовательно, не будет засчитано. Чем меньше цель, тем экспоненциально больше повторных бросков понадобится, чтобы выиграть. В конце концов, когда цель будет равна 2 (минимально возможная), только один бросок из каждых 36, или 2% из них, дадут победный результат.

В `<<sha256_example_generator_output>` > выигрышный "nonce" — это число 13 и этот результат может быть подтвержден независимо кем угодно. Любой пользователь может добавить номер 13 в виде суффикса к фразе "I am Satoshi Nakamoto" и вычислить хэш, убедившись, что он меньше цели. Успешный результат также является доказательством работы, потому что это доказывает, что мы проделали достаточно работы по нахождению nonce. В то время как для проверки требуется вычислить только один хэш, для того, чтобы найти подходящий nonce, нам потребовалось проделать 13 вычислений хэша. Если бы у нас была более низкая цель (большая сложность), то на нахождение nonce потребовалось бы гораздо больше вычислений, но все еще одно вычисление требовалось бы для проверки. Кроме того, зная цель, любой желающий может оценить сложность с помощью статистики и, следовательно, может оценить как много работы следует проделать для того, чтобы найти подобный nonce.

Доказательство работы Bitcoin очень похоже на изображенное на схеме [SHA256 Вывод скрипта генерации множества хэшей путем итерирования nonce](#). Майнер строит блок-кандидат, заполненный транзакциями. Затем майнер вычисляет хэш заголовка этого блока и проверяет не меньше он текущей *цели*. Если хэш не меньше цели, майнер меняет nonce (обычно просто увеличит его на один) и повторит попытку. При текущей сложности сети Bitcoin, майнерам требуется квадриллионы попыток, прежде чем будет найден nonce, который даст достаточно маленький хэш заголовка блока.

Очень упрощенный алгоритм доказательства работы на языке Python показан в [Упрощенная реализация доказательства работы](#).

Example 11. Упрощенная реализация доказательства работы

```
#!/usr/bin/env python
# example of proof-of-work algorithm

import hashlib
import time
```

```

max_nonce = 2 ** 32 # 4 billion

def proof_of_work(header, difficulty_bits):

    # calculate the difficulty target
    target = 2 ** (256-difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()

        # check if this is a valid result, below the target
        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 = ''

    # difficulty from 0 to 31 bits
    for difficulty_bits in xrange(32):

        difficulty = 2 ** difficulty_bits
        print "Difficulty: %ld (%d bits)" % (difficulty, difficulty_bits)

        print "Starting search..."

        # checkpoint the current time
        start_time = time.time()

        # make a new block which includes the hash from the previous block
        # we fake a block of transactions - just a string
        new_block = 'test block with transactions' + hash_result

        # find a valid nonce for the new block
        (hash_result, nonce) = proof_of_work(new_block, difficulty_bits)

        # checkpoint how long it took to find a result
        end_time = time.time()

        elapsed_time = end_time - start_time
        print "Elapsed Time: %.4f seconds" % elapsed_time

```

```
if elapsed_time > 0:

    # estimate the hashes per second
    hash_power = float(long nonce)/elapsed_time
    print "Hashing Power: %ld hashes per second" % hash_power
```

При запуске кода на выполнение, вы можете установить требуемую сложность (в битах, сколько из ведущих бит должны быть равны нулю) и посмотреть, сколько времени вашего компьютера потребуется для поиска решения. В [Пример работы алгоритма доказательства работы для различных уровней сложности](#), вы можете увидеть, как это выглядит на среднем ноутбуке.

Example 12. Пример работы алгоритма доказательства работы для различных уровней 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 029ae6e5004302a120630adcb808452346ab1cf0b94c5189ba8bac1d47e7903
```

```
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
```

Как можно заметить, увеличение сложности на 1 бит приводит к экспоненциальному увеличению времени, которое требуется на поиск решения. Зафиксировав значение всего одного бита в 256-битном числовом пространстве, вы уменьшите пространство поиска на половину. В примере [Пример работы алгоритма доказательства работы для различных уровней сложности](#) для поиска nonce, которое даст хеш с 26-тью ведущими битами, установленными в ноль, понадобится перебрать 84 миллионов хешей. Даже при скорости 120000 хэшей в секунду, на обычном ноутбуке на поиски понадобится 10 минут.

На момент написания, сеть находилась в состоянии поиска блока, хеш заголовка которого был бы меньше, чем 000000000000004c296e6376db3a241271f43fd3f5de7ba18986e517a243baa7. Как можно заметить, в начале этого хеша находится много нулей, а это означает, что допустимый диапазон хэшей намного меньше, следовательно, найти правильный хеш сложнее. Для обнаружения следующего блока понадобится в среднем более 150 квадриллионов вычислений хешей в секунду для сети. Это кажется невыполнимой задачей, но, к счастью мощность сети составляет 100 петахешей в секунду (PH/сек), что поможет найти блок в среднем примерно за 10 минут.

Представление сложности

В Блок 277,316, мы видели, что в блоке содержится целевая сложность, обозначаемая "биты сложности" или просто "биты" и которая для блока 277,316 содержит значение 0x1903a30c. Эта запись выражает цель сложности в формате коэффициент/показатель степени, с первыми двумя шестнадцатеричными цифрами для показателя степени и следующих шести шестнадцатеричных цифр в качестве коэффициента. В этом блоке показатель степени равен 0x19, а коэффициент 0x03a30c.

Формула для расчета целевой сложности из этого представления:

```
target = coefficient * 2^(8 * (exponent - 3))
```

Используя эту формулу, подставив в качестве сложности значение 0x1903a30c, мы получаем:

```
target = 0x03a30c * 2^(0x08 * (0x19 - 0x03))^
```

$$\Rightarrow \text{Цель} = 0x03a30c * 2 \wedge (0x08 * 0x16) \wedge$$

$$\Rightarrow \text{Цель} = 0x03a30c * 2^{\wedge} 0xB0^{\wedge}$$

что в десятичной системе счисления равняется:

```
=> 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
```

если переключиться обратно в шестнадцатеричную форму:

```
=> target = 0x00000000000000003A30C0000000000000000000000000000000000000000000
```

Это означает, что действительный блок для высоты 277316 — это такой, хеш заголовка которого меньше цели. В двоичном представлении этого числа больше 60-ти первых битов установлены в ноль. При подобном уровне сложности, одиночный майнер, находящий 1 триллион хэшей в секунду (1 тера-хэш в секунду или 1 ТН/сек) смог бы найти решение лишь один раз каждые 8496 блоков или в среднем один раз каждые 59 дней.

Целевая Сложность и Перенацеливание

Как мы видели ранее, цель определяет сложность и, следовательно, влияет на то, сколько времени потребуется алгоритму доказательства работы на поиск решения. Это приводит к

очевидным вопросам: почему сложность регулируется, кто и каким образом ее регулирует?

Блоки Bitcoin генерируются в среднем каждые 10 минут. Это сердцебиение Bitcoin задает частоту эмитирования валюты и скорость совершения транзакций. Оно должно оставаться постоянным не только в краткосрочной перспективе, но в течение многих десятилетий. За это время, ожидается, что мощность компьютеров будет продолжать расти быстрыми темпами. Кроме того, количество участников майнинга и компьютеры, которые они используют также будет постоянно меняться. Для того, чтобы сохранить время генерации блока в 10 минут, сложность майнинга должна быть скорректирована с учетом этих изменений. На самом деле, сложность представляет собой динамический параметр, который будет периодически корректироваться для достижения цели в "1 блок каждые 10-минут". Проще говоря, целевая сложность устанавливается так, что для майнинга любой мощности, интервал между блоками будет всегда 10 минут.

Каким же образом подобная корректировка выполняется в полностью децентрализованной сети? Перепланирование уровня сложности происходит автоматически и на каждом полном узле независимо друг от друга. Каждые 2016 блоков, все узлы перенацеливают сложность алгоритма доказательства работы. Уравнение для перенацеливания сложности измеряет время, которое потребовалось, чтобы найти последние 2016 блоков и сравнивает его с ожидаемыми 20160 минутами (две недели 10-минутных блоков). Полученное соотношение между фактическим и желаемым временными интервалами используется для корректировки сложности вверх или вниз. Проще говоря: если сеть находит блоки быстрее, чем через каждые 10 минут, сложность возрастает, если же блок находится медленнее, чем ожидалось, то сложность уменьшается.

Уравнение можно обобщить следующим образом:

$$\text{Новая Сложность} = \text{Старая Сложность} * (\text{Фактическое Время последних 2016 блоков} / 20160 \text{ минут})$$

В [Перевычисление сложности доказательства работы — CalculateNextWorkRequired\(\)](#) в `pow.cpp` показан код, используемый в клиенте Bitcoin Core.

Example 13. Перевычисление сложности доказательства работы — *CalculateNextWorkRequired()* в *pow.cpp*

```
// Ограничиваем шаг настройки
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;

// Перенастройка
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;
```

NOTE

В то время как калибровка сложности происходит каждые 2016 блока, из-за ошибки в оригинальном клиенте Bitcoin Core, она базируется на общем времени предыдущих 2015 блоков (а не 2016, как это должно быть), что приводит к смещению в сторону более высокой сложности на 0.05%.

Параметры Interval (2016 блока) и TargetTimespan (две недели в виде 1209600 секунд) определены в *chainparams.cpp*.

Для того, чтобы избежать резких скачков сложности, регулировка перенацеливания должна быть меньше, чем в четыре раза за один цикл. Если требуется регулирование сложности больше, чем в четыре раза, то будет взят максимум и не более. Любая дальнейшая корректировка будет осуществляться в следующем периоде перенацеливания, поскольку дисбаланс будет сохраняться в течение следующих 2016 блоков. Таким образом, балансировка больших расхождений между мощностью хеширования и сложностью может занять несколько циклов 2016 блоков.

ТИП

Сложность поиска блока составляет около '10 минут обработки' для всей сети, на основании времени, которое потребовалось, чтобы найти предыдущие 2016 блока, при уточнении каждые 2016 блока.

Обратите внимание, что целевая сложность не зависит от количества транзакций или их

величины. Это означает, что мощность хэширования и, следовательно, количество электричества, затраченного для обеспечения безопасности Bitcoin также совершенно не зависит от количества транзакций. Bitcoin может масштабироваться, получать более широкое внедрение, и оставаться безопасным без какого-либо увеличения мощности хэширования относительно сегодняшнего уровня. Увеличение мощности хеширования представляет рыночные силы по мере входа новых майнеров на рынок, которые конкурируют за вознаграждение. Пока под контролем майнеров действующих честно в погоне за наградой находится достаточно мощностей хеширования для предотвращения атак «перехвата» и, следовательно, достаточно, чтобы обеспечить безопасность Bitcoin.

Целевая сложность тесно связана со стоимостью электроэнергии и обменным курсом Bitcoin. Высокоэффективные майнинговые системы преобразовывают электричества в вычисления хешей на максимально возможной скорости. Основное влияние на рынке майнинга оказывает цена одного киловатт-часа в биткоинах, т.к. это определяет рентабельность майнинга и, следовательно, стимулы для входа или выхода из рынка.

Удачно найденный блок

Как мы видели ранее, узел Цзиня построил блок-кандидат и подготовил его майнинга. Цзинь имеет несколько майнинговых ферм на основе ASIC, где сотни тысяч интегральных схем прогоняют алгоритм SHA256 параллельно с невероятной скоростью. Эти специализированные компьютеры подключены к его майнинговому узлу через USB. Далее, майнинговый узел, работающий на рабочем компьютере Цзиня передает заголовок блока к майнинговому оборудованию, которое начинает тестирование триллионов nonce в секунду.

Почти через 11 минут после начала поиска блока 277316, одна из майнинговых машин находит решение и отправляет его обратно в майнинговый узел. Nonce равный 4215469401 в заголовке блока, дает хеш блока:

00000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569

который меньше, чем цель:

00000000000000003A30C000

Майнинговый узел Цзиня немедленно передает блок всем своим пирам. Те получают, валидируют и передают дальше новый блок. По мере того как блок распространяется по сети, каждый узел добавляет его в собственную копию blockchain, расширив его до новой высоты в 277316 блоков. Если майнинговый узел получает и валидирует блок, он отказывается от попыток найти блок на той же высоте и сразу же начинает вычисления следующего блока в цепочке.

В следующем разделе мы рассмотрим процесс, который каждый узел использует для

валидации блока и выбора самую длинную цепочки, создавая консенсус, лежащий в основе децентрализованного blockchain.

Валидация нового блока

Третий шаг механизма консенсуса Bitcoin — это независимая валидация каждого нового блока каждым узлом в сети. По мере распространения каждого нового решенного блока по сети, каждый узел выполняет серию тестов по его проверке, прежде чем распространить его дальше своим пирам. Это гарантирует, что только валидные блоки будут распространяться по сети. Независимая проверка также гарантирует, что блоки, найденные честными майнерами, будут включены в blockchain и тем самым заработают им награду. Блоки тех майнеров, которые будут действовать нечестно, будут отвергнуты и не только лишат их награды, но и гарантируют, что усилия, затраченные на поиск решения, пройдут впустую, принеся убытки в виде счета за электроэнергию.

Когда узел получает новый блок, он проверяет его по длинному списку критериев, в случае неудовлетворения которых блок отвергается. Эти критерии можно увидеть в клиенте Bitcoin Core в следующих функциях CheckBlock и CheckBlockHeader. Проверки включают в себя:

- Структура блока данных синтаксически валидна
- Хэш заголовка блока меньше, чем целевая сложность
- Временная метка блока не убегает более чем два часа в будущее
- Размер блока находится в допустимых пределах
- Первая транзакция (и только первая) — это coinbase-транзакция
- Все транзакции внутри блока проходят валидацию, описанную в [Независимая проверка сделок](#)

Независимая валидация каждого нового блока каждым узлом сети гарантирует, что майнеры не смогут обманывать. В предыдущих разделах мы видели, как майнеры создают транзакцию, которая назначает им новые биткойны, созданные внутри блока и комиссионные за транзакции. Почему майнеры не могут "нарисовать" сами себе транзакцию на тысячу биткойнов вместо правильного вознаграждения? Потому, что каждый узел проверяет блоки в соответствии с одними и теми же правилами. Недействительная coinbase-транзакция сделает весь блок недействительным, что приведет к отвержению блока и, следовательно, эта транзакция никогда не станет частью бухгалтерской книги. Майнеры должны построить идеальный блок на основании общих правил, которым следуют все узлы, и находить его с правильным решением доказательства работы. Для достижения этого, они расходуют много электричества во время майнинга, и если они будут обманывать, все это электричество и все усилия пропадут впустую. Поэтому независимая проверка является ключевым компонентом децентрализованного консенсуса.

Сборка и выбор цепочки блоков

Заключительный шаг в механизма децентрализованного консенсуса Bitcoin является сборка блоков в цепочки и выбор цепи с наибольшим доказательством работы. После того, как узел подтвердил новый блок, он попытается собрать цепочку, подключив блок к существующему blockchain.

Узлы поддерживают три набора блоков: те, которые связаны с главным blockchain, те, которые образуют ответвления от главного blockchain (вторичные цепи) и, наконец, блоки, которые не имеют известного родителя в известных цепочках (бесхозные блоки). Недействительные блоки отвергаются как только любой один из критериев проверки не проходит, и поэтому они не включаются в какую-либо цепь.

"Основная цепочка" в любой момент времени — это та, которая содержит в себе наибольшую совокупную сложность. В большинстве случаев это также цепь с наибольшим количеством блоков, если только не существует двух, имеющих одинаковую длину и тогда одна из содержит больше доказательства работы. Основная цепь также будет иметь ответвления с блоками, которые будут "братьями" блокам из основной цепи. Эти блоки будут валидными, но не являться частью основной цепи. Они будут отложены на случай, если в будущем одна из этих цепей будет удлинена до превышения сложности основной цепи. В следующем разделе ([Форки блокчейна](#)), мы увидим, как вторичные цепи появляются как результат почти одновременного обнаружения блоков на одной и той же высоте.

После получения нового блока, узел попытается встроить его в существующий blockchain. Узел прочитает поле "хэш предыдущего блока", в котором хранится ссылка на родительский блок, а затем попытается найти этот блок в blockchain. Чаще всего родитель будет последним элементом основной цепи, что будет означать, что этот новый блок расширяет основную цепь. Например, новый блок 277316 содержит ссылку на хэш его родительского блока номер 277315. Большинство узлов, получивших 277316 уже будут иметь у себя блок 277315 в конце основной цепи и, следовательно, привяжут новый блок и расширят эту цепь.

Иногда, как мы увидим в [Форки блокчейна](#), новый блок расширяет цепь, которая не является основной. В этом случае узел присоединит новый блок ко вторичной цепи, а затем сравнит сложность вторичной цепи с основной. Если вторичная цепь имеет больше накопленной сложности, узел будет *повторит сходимост* на вторичной цепи, то есть он выберет вторичную цепь в качестве новой главной цепи, что делает старую основную цепь вторичной цепью. Если узел майнинговый, он построит блок, расширяющий эту новую, более длинную цепь.

Если был получен валидный блок и ни один родитель не найден в существующих цепях, этот блок считается «сиротой». Блоки-сироты хранятся в пуле сиротских блоков, где они продолжают находиться пока не будут получены их родители. После того, как родительский блок оказывается получен и связан в существующей цепью, блок-сирота может быть вытащен из сиротского пула и связан с родителем, что сделает его частью цепи. Блоки-сироты, как правило, возникают когда два блока, добытых в течение короткого промежутка времени друг

от друга бывают получены в обратном порядке (дочерний до родительского).

Выбрав цепь с наибольшей сложностью, все узлы в конечном итоге достигают общего консенсуса сети. Временные расхождения между цепями также в конце концов разрешаются вместе с расширением одной из возможных цепочек. Майнинговые узлы "голосуют" своими мощностями, выбирая, какую из цепочек расширить путем нахождения следующего блока. Когда они находят новый блок и расширяют цепочку, сам новый блок представляет их голос.

В следующем разделе мы рассмотрим, как расхождения между конкурирующими цепями (форками) разрешаются путем независимого выбора цепи самой длинной сложности.

Форки блокчейна

Поскольку блокчейн представляет собой децентрализованную структуру данных, различные его копии не всегда согласованы. Блоки могут прийти к различным узлам в разное время, в результате чего узлы будут иметь различающееся видение blockchain. Чтобы решить эту проблему, каждый узел всегда выбирает и пытается удлинить ту цепочку блоков, которая представляет большее доказательство работы, также известную как длиннейшая цепь или цепь с наибольшей совокупной сложностью. Суммируя сложность, записанную в каждом блоке цепи, узел может вычислить общее количество работы, которая была проведена для создания данной цепи. До тех пор пока все узлы выбирают длиннейшую по совокупной сложности цепочку, глобальная сеть Bitcoin в конечном итоге сходится к согласованному состоянию. Форки случаются вследствие временных расхождений между версиями blockchain, которые в конце концов разрешаются по мере добавления блоков к одному из форков.

На следующих нескольких диаграммах, мы проследим за распространением события «форка» по сети. Диаграмма представляет собой упрощенное представление Bitcoin в качестве глобальной сети. В действительности, топология Bitcoin сети не организована географически. Скорее, она образует собой меш-сеть соединенных между собой узлов, которые могут быть расположены очень далеко друг от друга географически. Представление географической топологии является упрощенным и используется в целях иллюстрации форка. В реальной сети Bitcoin, "расстояние" между узлами измеряется в "хопах" от узла к узлу, а не на основе их физического местоположения. В целях иллюстрации, различные блоки отображаются различными цветами, по мере распространения по сети окраскивающих соединения, которые они пересекают.

На первой диаграмме ([Визуализация события раздвоения blockchain—до возникновения форка](#)), сеть имеет единую перспективу blockchain, с синим блоком на вершине основной цепи.

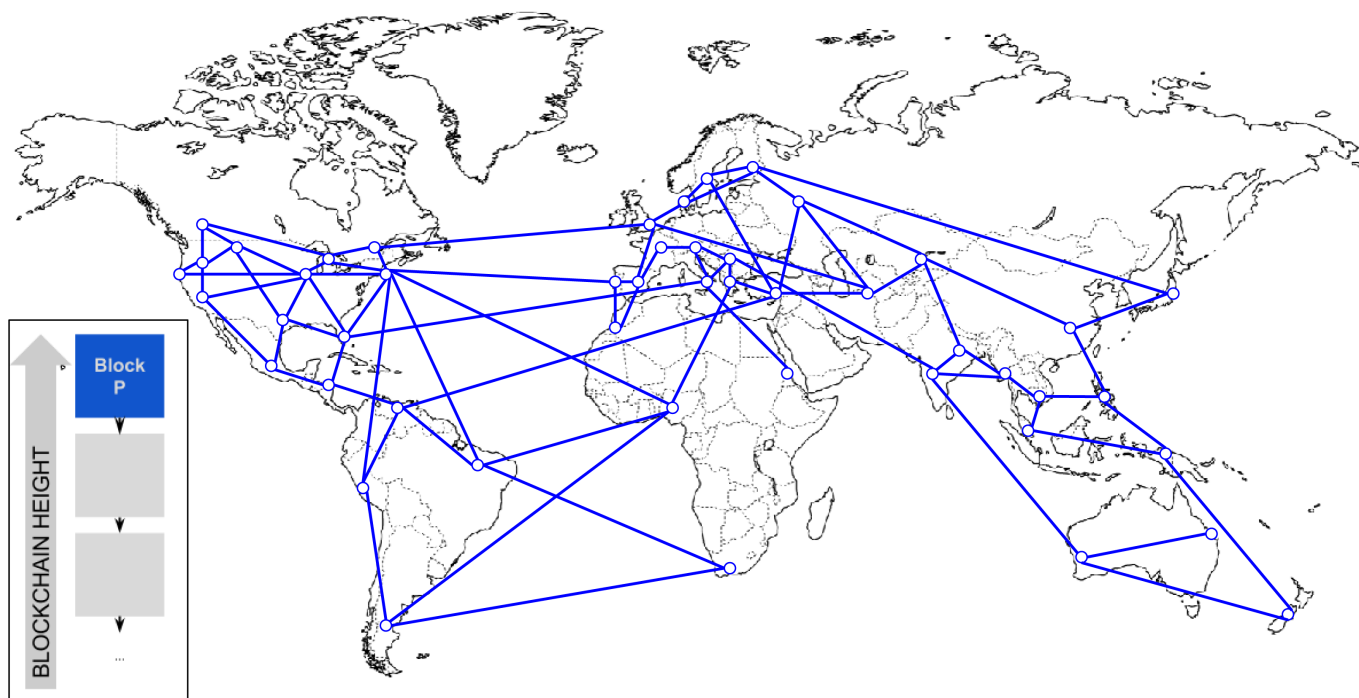


Figure 2. Визуализация события раздвоения blockchain—до возникновения форка

"Форк" происходит всякий раз, когда есть два конкурирующих блока кандидата на формирование более длинной blockchain. Это происходит в нормальных условиях, когда два майнера находят решение алгоритма доказательства работы в течение короткого периода времени относительно друг от друга. Поскольку оба майнера находят решение для соответствующих блоков-кандидатов, они сразу транслируют свой собственный "выигрышный" блок своим ближайшим пирам, которые начинают распространять блок по сети. Каждый узел, получивший валидный блок, включит его в свою копию blockchain, удлинив его на один блок. Если этот узел позже видит другой блок-кандидат, расширяющий один и тот же родительский, он присоединяет второго кандидата ко вторичной цепи. В результате, некоторые узлы будут "видеть" сначала один блок-кандидат, в то время как другие узлы будут видеть другой блок-кандидат и возникнет две конкурирующие версии blockchain.

В Визуализация раздвоения blockchain: два блока найдено одновременно, мы видим двух майнеров, которые находят два разных блока почти одновременно. Оба этих блока являются потомками синего блока, один из них, найденный в Канаде, визуализируется красным цветом, а другой обозначен как зеленый блок и найден в Австралии.

Предположим, например, что майнер в Канаде находит решение доказательства работы для "красного" блока, который расширяет blockchain, опираясь на "синий" родительский блок. Почти одновременно с ним, австралийский майнер, который также расширял "синий" блок находит решение для "зеленого" блока, своего блока-кандидата. Теперь, имеется два возможных блока, один, происходящий из Канады мы называем «красный», и тот, который мы называем "зеленый", происходящий из Австралии. Оба блока валидны, оба содержат допустимое решение доказательства работы, и оба блока расширяют блокчейн от общего родителя. Оба блока, вероятно, содержат по большей части одни и те же транзакции, только возможно с несколькими различиями в порядке транзакций.

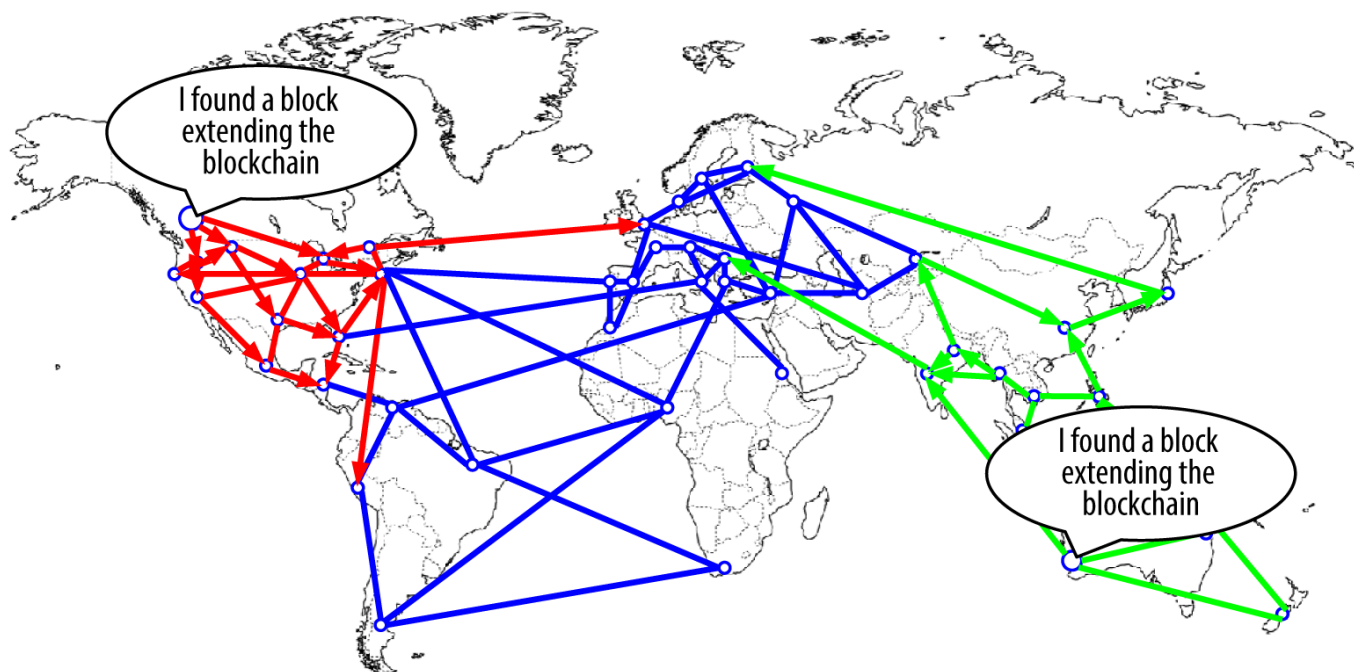


Figure 3. Визуализация раздвоения blockchain: два блока найдено одновременно

По мере распространения обоих блоков, некоторые узлы получают "красный" блок первым, а некоторые "зеленый". Как показано в [Визуализация раздвоения blockchain: распространение двух блоков, разделяющее сеть пополам](#), сеть разделяется на две различные точки зрения на blockchain, с красным блоком и с зеленым блоком на вершинах.

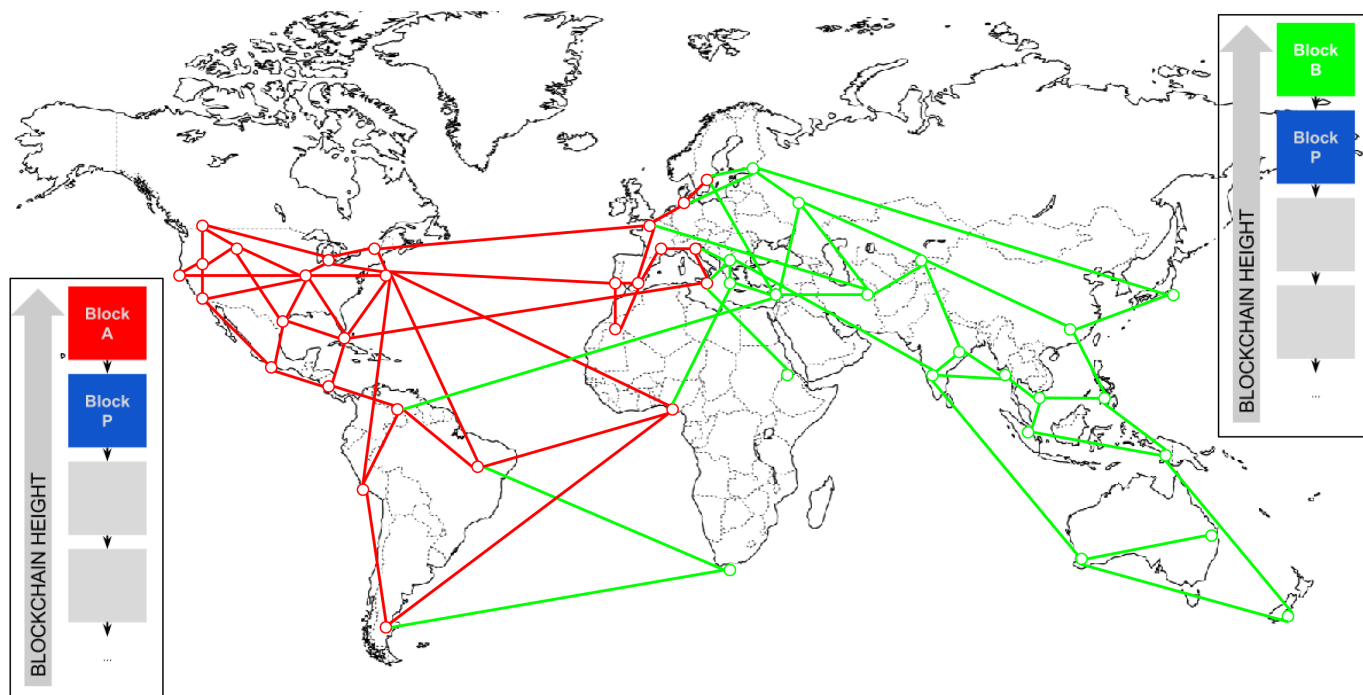


Figure 4. Визуализация раздвоения blockchain: распространение двух блоков, разделяющее сеть пополам

С этого момента, узлы сети, находящийся ближе всех (топологически, а не географически) к канадскому узлу узнают о "красном" блоке первыми и создадут новый блокчейн с-самой-

большой-накопленной-сложностью с "красным" блоком в качестве последнего в цепочке (например, синий-красный), не обращая внимания на "зеленый" блок-кандидат, который прибудет немного позже. В то же время, узлы ближе к австралийскому узлу будут считать австралийский блок победителем и удлинят блокчейн "зеленым" блоком в качестве последнего (например, синий-зеленый), не обращая внимания на "красный", когда тот поступит через несколько секунд. Любые майнеры, которые видели "красный" первым сразу же построят блоки-кандидаты, ссылающиеся на "красный" блок в качестве родителя и начнут искать решение доказательства работы для этих блоков-кандидатов. Майнеры, которые приняли «зеленый» вместо начнут создание блока поверх "зеленого" и расширят эту цепь.

Форки как правило всегда разрешаются в пределах одного блока. Как часть мощности хэширования сети уходит на построение на вершине "красного" родителя, другая часть мощности хэширования сфокусирована на строительстве блока на вершине "зеленого" блока. Даже если мощность хэширования разделится почти поровну, вполне вероятно, что один набор майнеров найдет решение и распространит его до того, как другое множество майнеров обнаружит какое-либо решение. Скажем, например, что майнеры, строящие на вершине "зеленого" блока, нашли новый "розовый" блок, который расширяет цепь (например, синий-зеленый-розовый). Они сразу же распространяют этот новый блок и вся сеть видит его в качестве действительного решения, как показано на [Визуализация раздвоения blockchain: новый блок удлиняет один из форков](#).

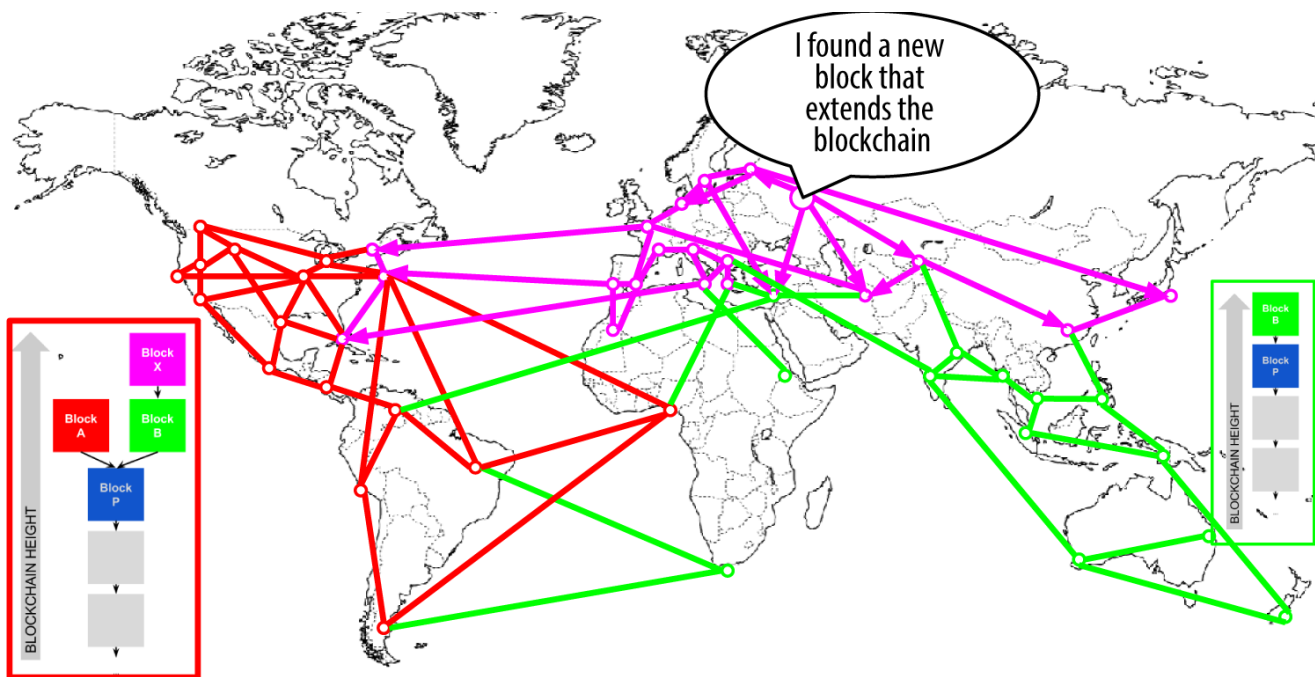


Figure 5. Визуализация раздвоения blockchain: новый блок удлиняет один из форков

Все узлы, которые выбрали "зеленый" в качестве победителя в предыдущем туре, просто продлят цепочку еще один блок. Узлы, которые выбрали "красный" в качестве победителя, теперь будут видеть две цепи: синий-зеленый-розовый и синий-красный. Цепь синий-зеленый-розовый теперь длиннее (содержит больше совокупной сложности), чем цепь синий-красный. В результате, эти узлы примут цепь синий-зеленый-розовый в качестве основной

цепи, а цепь синий-красный станет вторичной, как показано в [Визуализация раздвоения blockchain: сеть сходится на новой длиннейшей цепи](#). Эти узлы будут вынуждены пересмотреть свой взгляд на блокчейн из-за появления новых доказательств более длинной цепи. Любые майнеры, работающие над расширением цепи синий-красный прекратят эту работу, потому что их блок-кандидат станет "сиротой", так как его родитель "красный" больше не находится в самой длинной цепи. Транзакции, содержащиеся в "красном" блоке будут снова поставлены в очередь для обработки в следующем блоке, т.к. этот блок уже не будет в основной цепи. Вся сеть повторно сойдется на единой цепочке синий-зеленый-розовый, с "розовым" в качестве последнего. Все майнеры немедленно начнут работу над блоками-кандидатами, которые будут ссылаться на "розовый" в качестве своего родителя.

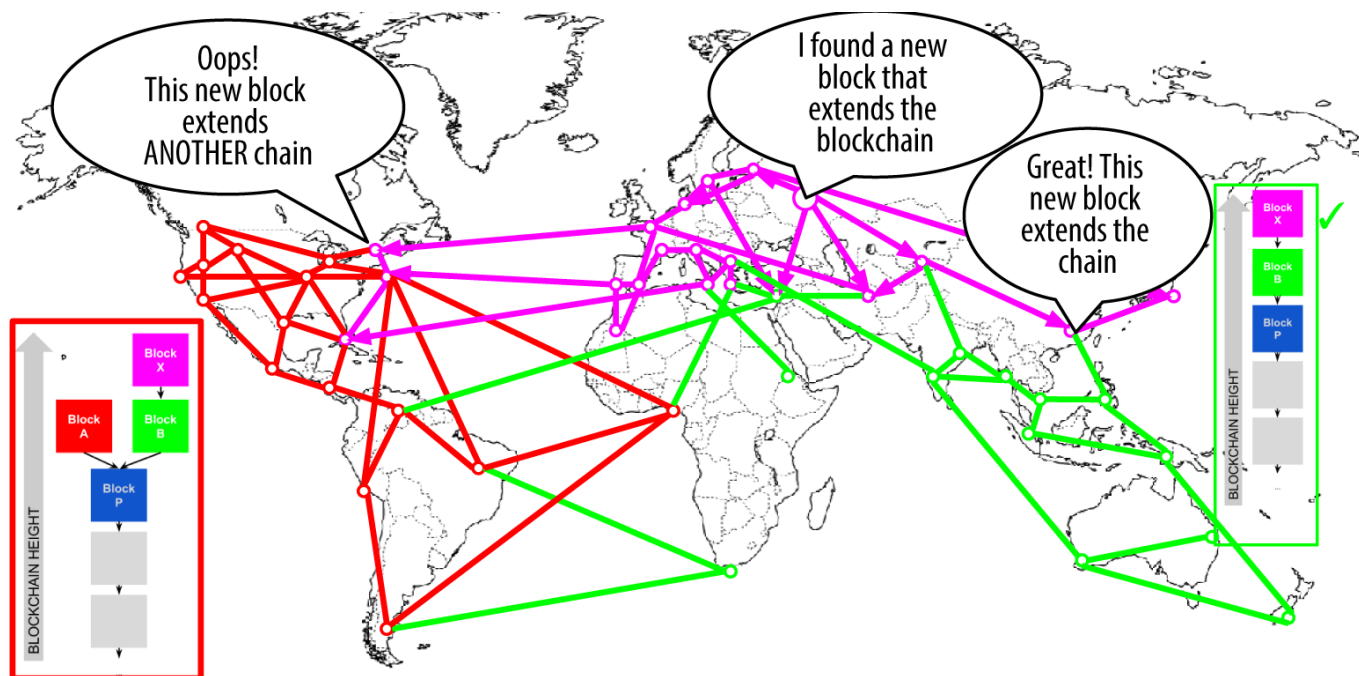


Figure 6. Визуализация раздвоения blockchain: сеть сходится на новой длиннейшей цепи

Теоретически возможно, что форк распространится на два блока, если два блока обнаружены почти одновременно майнерами на противоположных "сторонах" предыдущего форка. Тем не менее, вероятность такого события очень мала. В то время одноблочный форк может происходить каждую неделю, двублочный форк чрезвычайно редок.

Интервал между блоками Bitcoin длиной 10 минут является компромиссом между быстрым временем подтверждения и вероятностью возникновения форка. Более короткий интервал сделал бы взаиморасчеты быстрее, но привел бы к более частым форкам blockchain, в то время как больший интервал уменьшил бы количество форков, но сделал бы взаиморасчеты медленнее.

Майнинг и гонка и хеширования

Bitcoin-майнинг — это чрезвычайно конкурентная индустрия. Мощности хеширования росли экспоненциально каждый следующий год существования Bitcoin. В некоторые годы рост

выразился в полное изменение технологии, например, в 2010 и 2011 годах многие майнеры перешли с использования ЦП на майнинг с помощью GPU и FPGA. В 2013 году появление ASIC-майнинга, где функция SHA256 воплощалась непосредственно в виде кремниевых чипов, привело к следующему гигантскому скачку мощности хеширования. Первые подобные чипы имели больше мощностей хеширования, чем вся Bitcoin-сеть в 2010 году.

Общая мощность хеширования сети Биткоин, в течение первых пяти лет:

2009

0.5 MH/sec–8 MH/sec (рост 16%)

2010

8 MH/sec–116 GH/sec (рост 14,500%)

2011

16 GH/sec–9 TH/sec (рост 562%)

2012

9 TH/sec–23 TH/sec (рост 2.5%)

2013

23 TH/sec–10 PH/sec (рост 450%)

2014

10 PH/sec–150 PH/sec in August (рост 15%)

На графике [Общая мощность хэширования, в гигахэшах в секунду, в течение двух лет](#) изображено увеличение мощности хэширования сети Bitcoin за последние два года. Как можно видеть, конкуренция между майнерами и рост Bitcoin привел к экспоненциальному росту мощности хеширования (всего хэшей в секунду по всей сети).



Figure 7. Общая мощность хэширования, в гигахэшах в секунду, в течение двух лет

Так как мощность хэширования резко увеличилась, сложность выросла соответственно. Сложность показана на [Сложность майнинга Биткоин в течение двух лет](#) и измеряется как отношение текущей сложности к минимальной сложности (сложность первого блока).

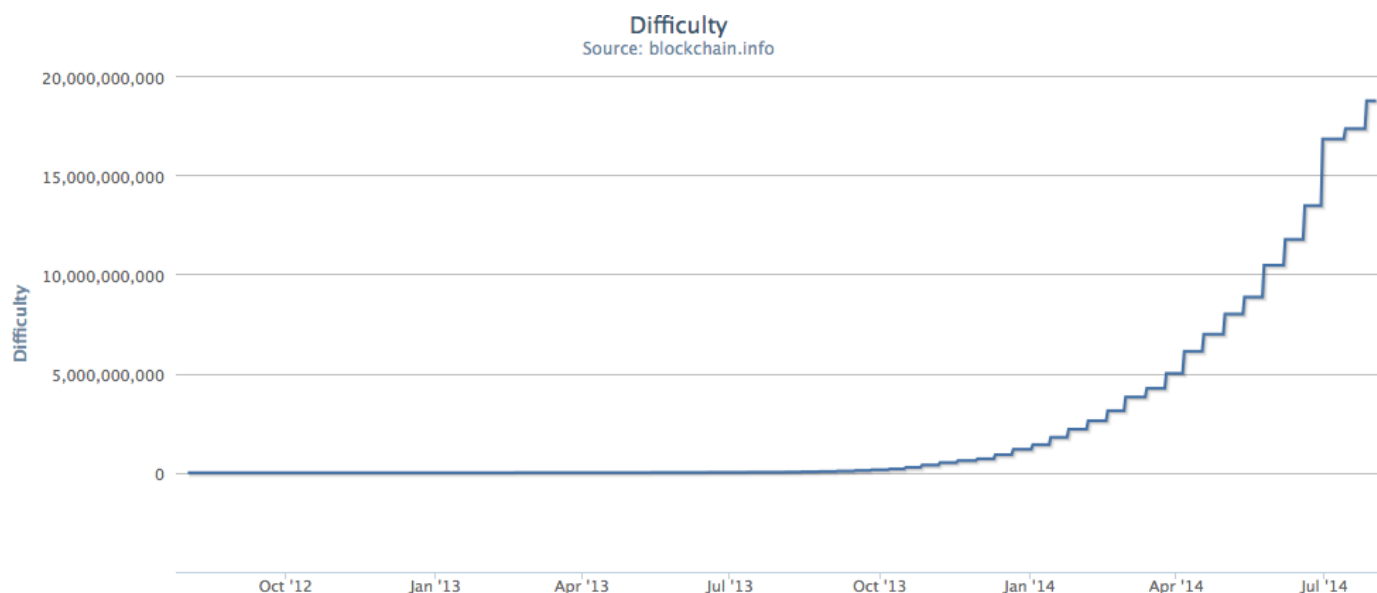


Figure 8. Сложность майнинга Биткоин в течение двух лет

За последние два года, майнинговые ASIC-чипы становятся все более совершенными, приближаясь к границам возможностей технологии изготовления полупроводниковых чипов по 22 нанометровому техпроцессу. В настоящее время производители ASIC-чипов стремятся обогнать производителей чипов общего назначения, проектируя чипы по 16 нанометровому техпроцессу, поскольку рентабельность майнинга толкает отрасль даже быстрее, чем потребность в процессорах общего назначения. Майнинг больше не сможет далее идти семимильными шагами, так как отрасль упирается в закон Мура, который гласит, что вычислительная мощность компьютеров удваивается примерно каждые 18 месяцев. Тем не менее, мощность сети продолжает увеличиваться экспоненциально благодаря непрекращающейся гонке за более высокими плотностями транзисторов и росту размеров дата-центров. Речь идет уже не о мощностях майнинга одного чипа, но о том, сколько этих чипов может поместиться в одном помещении и как при этом обеспечить отвод тепла и достаточное количество электроэнергии.

Решение Extra Nonce

С 2012 года биткоин-майнинг уперся в фундаментальное ограничение в структуре заголовка блока. В первые дни Bitcoin, майнер мог найти блок перебирая nonce, пока полученный хэш не оказывался ниже целевого значения. По мере увеличения сложности, майнеры часто циклично проходили через все 4 миллиарда значений nonce без обнаружения блока. Тем не менее, эта проблема легко решалась путем обновления метки времени блока с учетом затраченного времени. Так как временная метка является частью заголовка, это изменение позволяло майнерам пройти новую итерацию значения nonce с другими результатами. Когда

мощность майнингового оборудования превысила 4 GH/сек это стало проблемой, поскольку все значения nonce исчерпывались менее чем за секунду. По мере того, как ASIC-чипы начали достигли мощностей свыше TH/сек, программное обеспечение для майнинга потребовало больше места под nonce. Метка времени может быть немного отложена, но значение где-то слишком далеко в будущем приведет к объявлению блока недействительным. В заголовке блока требовалось какое-то новое поле для этих целей. Выходом стало использовать coinbase-транзакцию в для хранения дополнительных значений nonce. Поскольку coinbase сценарий может хранить от 2 до 100 байт данных, майнеры начали использовать это неиспользуемое место в качестве дополнительного nonce. Coinbase транзакция включается в дерево Меркле, что означает, что любое изменение в coinbase сценарии вызывает изменение корня дерева. Восемь байт дополнительного nonce, плюс 4 байта "стандартного" nonce позволяют майнерам проверить всего 2^{96} (8 с 28 нулями) значений *в секунду* без необходимости изменять метку времени. Если в будущем, майнерам и этого окажется мало, они могут как и раньше обновить временную метку. Кроме того, в coinbase-сценарии еще достаточно места для расширения nonce в будущем.

Майнинговые пулы

В этой высоко конкурентной среде, индивидуальные майнеры практически не имеет шансов. Вероятность самостоятельного нахождения блока для компенсации затрат на электроэнергию и оборудование настолько мала, что майнинг становится больше похож на игру в лотерею. Даже самая быстрая система потребительская ASIC-аппаратура не может идти в ногу с коммерческими системами, которые состоят из десятков тысяч этих чипов внутри гигантских складов вблизи гидро-электростанций. Майнеры в настоящее время собираются в пулы тысячами с целью объединения мощностей. Участвуя в пуле, майнеры получают небольшую долю от общего вознаграждения, но, как правило, получают это вознаграждение каждый день, уменьшая риски.

Давайте рассмотрим конкретный пример. Предположим, майнер приобрел оборудования с общей мощностью хеширования 6000 гигахешей в секунду (GH/s) или 6 TH/s. В августе 2014 года это оборудование стоило приблизительно \$10000. Оборудование потребляет 3 кВт (kW) электроэнергии при работе, 72 кВт-часов в сутки, по цене \$7 или \$8 в день в среднем. При текущей сложности Bitcoin, соломайнер сможет находить один блок примерно раз в 155 дней, или каждые 5 месяцев. Если майнер находит блок в указанный срок, он получает 25 биткоинов, по цене примерно \$600 за единицу, даст в сумме \$15000, что покроет всю стоимость оборудования и электроэнергии, потребленной в течение этого периода времени, оставив чистую прибыль в размере около \$3000. Тем не менее, вероятность найти блок в пятимесячный период зависит от удачи майнера. Он мог бы найти два блока в течение пяти месяцев и получить очень большую прибыль. Или он может не найти блок в течение 10 месяцев и понесет финансовые потери. Хуже то, что с такими темпами роста сложность алгоритма доказательства работы, скорее всего, значительно вырастет в течение этого периода и у майнера в лучшем случае шесть месяцев чтобы выйти "по нулям" пока его оборудование фактически не устареет и должно будет заменено на более мощное. Если этот майнер выберет участвовать в майнинг-пуле, вместо того, чтобы ждать выигрыша \$15000 каждые пять месяцев, он сможет заработать примерно \$500-\$750 в неделю. Регулярные выплаты пула

помогут ему амортизировать стоимость аппаратуры и электроэнергии в течение долгого времени без лишнего риска. Аппаратное обеспечение скорее всего все же устареет через 6-9 месяцев, но в течение этого периода доход по крайней мере будет регулярным и стабильным.

Майнинговые пулы координируют многие сотни или даже тысячи майнеров посредством специализированных протоколов пул-майнинга. После создания учетной записи в пуле, отдельные майнеры настраивают свое оборудование для подключения к серверу пула. Их майнинговое железо остается подключенным к серверу пула во время майнинга, синхронизируя свои усилия с другими майнерами. Таким образом, майнеры в пулах разделяют усилия на поиск блока, а затем разделяют между собой награду.

Найденные блоки платят вознаграждение по адресу Bitcoin-пула, а не отдельных майнеров. Сервер пула будет периодически осуществлять платежи по адресам майнеров, как только их доля от награды достигает определенного порогового значения. Как правило, сервер пула взимает процентную плату от вознаграждений за предоставление услуг майнингового пула.

Майнеры, участвующие в пуле, разделяют работу по поиску решения для блока-кандидата, зарабатывая «доли» за их вклад в майнинг. Майнинговый пул устанавливает более низкую цель сложности за зарабатывание доли, как правило, более чем в 1000 раз легче сложности сети Bitcoin. Когда кто-то в пуле успешно находит блок, награду получает пул, а затем разделяет ее со всеми майнерами пропорционально количеству долей, которыми они внесли свой вклад.

Пулы открыты для любых майнеров, больших или маленьких, профессионалов или любителей. Поэтому в пуле будут участвовать люди с одной небольшой майнинговой системой, и другие люди с гаражом, полным майнинговой аппаратуры высшего класса. Некоторые из них будут майнить на несколько десятков киловатт электроэнергии, у других будет свой дата-центр, потребляющий мегаватты мощности. Каким образом майнинг пул измеряет индивидуальные вклады, с тем, чтобы была возможность справедливо распределять награды, без возможности обмана? Ответ на этот вопрос заключается в использовании алгоритма доказательства работы для оценки вклада каждого майнера, который настраивается на более низкую сложность, так что даже самые маленькие майнеры выигрывают свою долю достаточно часто, чтобы их вклад в общий пул стал заметен. Устанавливая более низкую сложность для заработка долей, пул измеряет объем работы, проделанной каждым майнером. Каждый раз, когда майнер пула находит хэш заголовка блока, меньше, чем сложность пула, он доказывает, что была проделана работа по хешированию для нахождения этого результата. Что еще более важно, работа по нахождению долей статистически измеримым образом способствует общим усилиям по нахождению хэша ниже целевого уровня Bitcoin сети. Тысячи майнеров, пытающихся найти хеши с маленьким значением в конечном итоге найдут один такой, который бы удовлетворил цель Bitcoin сети.

Вернемся к аналогии игры в кости. Если игроки бросают кости с целью выбросить меньше четырех (общая сложность сети), пул будет мог бы установить более простую цель, считая сколько раз игрокам из пула удалось бросить меньше восьми. Когда игроки из пула выбрасывают меньше, чем восемь (цель доли пула), они зарабатывают доли, но не выигрывают игру, потому что не достигают цели игры (меньше четырех). Игроки пула будут

достигать более легкой цели пула гораздо чаще, зарабатывая им доли с регулярностью, даже если они не достигают более трудной цели и выигрыша в игре. Рано или поздно один из игроков пула выбросит комбинированный бросок костей с числом меньше четырех и пул выиграет. Затем, доходы могут быть распределены в пуле игроков на основе долей, которые те заработали. Даже если цель восемь-или-менее не была выигрышной, это был справедливый способ измерения бросков костей игроков, и иногда он давал бросок меньше-чем-четыре.

Аналогичным образом, майнинг пул установит такую сложность пула, которая будет гарантировать, что индивидуальный майнер пула сможет найти хэши заголовка блока, которые будут меньше сложности довольно часто, зарабатывая доли. Время от времени одна из этих попыток даст хэш заголовка блока, который будет меньше целевого показателя Bitcoin сети, что сделает его действительным блоком и весь пул выиграет.

Управляемые пулы

Большинство майнинговых пулов "управляемые", что означает, что есть компания или частное лицо, владеющее сервером пула. Владелец сервера пула называется *оператором пула*, и он удерживает с майнеров пула процентную комиссию от заработка.

Сервер пула работает под управлением специализированного программного обеспечения и протокола пул-майнинга, который координирует деятельность майнеров пула. Сервер также подключен к одному или более полных узлов Bitcoin и имеет прямой доступ к полной копии базы данных blockchain. Это позволяет серверу пула валидировать блоки и транзакции от имени майнеров пула, позволяя им не держать у себя полный узел. Для майнеров, это является важным фактором, так как под полный узел необходим отдельный компьютер с по крайней мере 15-20 Гб дискового пространства и, по меньшей мере, 2 Гб оперативной памяти (RAM). Кроме того, программное обеспечение Bitcoin, работающее на полном узле необходимо поддерживать и обновлять. Любое время простоя, вызванное отсутствием технического обслуживания или отсутствия ресурсов повредит рентабельности майнера. Для многих майнеров, возможность майнить без требования держать запущенным полный узел является еще одним большим плюсом.

Пул-майнеры подключаются к серверу пула с помощью таких протоколов, как Stratum (STM) или GetBlockTemplate (GBT). Более старый стандарт под названием GetWork (GWK) был признан устаревшим с конца 2012 года, так как не поддерживал майнинг на скоростях выше 4 GH/s. Оба протокола STM и GBT создают *шаблоны* блока, которые содержат шаблон заголовка блока-кандидата. Сервер пула строит блок-кандидат путем объединения транзакций, добавления coinbase-транзакции (с дополнительным nonce), вычислив корень Меркле, и поставив ссылку на хэш предыдущего блока. Заголовок блока-кандидата затем посылается каждому из майнеров пула в качестве шаблона. Каждый майнер затем майнит с использованием этого шаблона на более низкой сложности, чем сложность сети Bitcoin и посылает любые успешные результаты обратно на сервер пула, чум зарабатывает себе доли.

P2Pool

Управляемые пулы открывают возможности для обмана операторами пула, которые могут

направлять усилия пула для атак на повторное расходование или аннулирования блоков (см [Атаки на консенсус](#)). Кроме того, централизованные серверы пула представляют собой единую точку отказа. Если сервер пула остановлен или под атакой на отказ в обслуживании, майнеры пула не могут майнить. В 2011 году, чтобы решить эти проблемы централизации, был предложен и реализован новый метод майнинга: пиринговый P2Pool без центрального оператора.

P2Pool работает посредством децентрализации функций сервера пула, с помощью параллельной blockchain-подобной система под названием *share chain*. Share chain — это blockchain с более низкой сложностью, чем основной blockchain. Share chain позволяет пулу майнеров работать скооперировано в децентрализованном пуле, зарабатывая одним долевым блоком каждые 30 секунд. Каждый блок в share chain регистрирует пропорциональное доленое вознаграждение для работающих майнеров пула, передавая доли дальше из предыдущего блока. Когда один из блоков также достигает целевой сложности сети Bitcoin, он распространяется и включается в блокчейн Биткойн, награждая всех майнеров пула, внесших свой вклад в доли, которые способствовали обнаружению блока. По сути, вместо сервера пула, учитывающего доли и вознаграждения, share chain позволяет всем майнерам пула следить за всеми долями при помощи механизма децентрализованного консенсуса подобного механизму консенсуса блокчейна Биткойн.

P2Pool-майнинг технически более сложен, чем пул-майнинг, поскольку он требует, чтобы майнеры имели выделенный компьютер с достаточным объемом дискового пространства, памяти и пропускной способности для поддержки полного узла Bitcoin и программного обеспечения узла P2Pool. P2Pool майнеры подсоединяют оборудование к своему локальному узлу P2Pool, который имитирует функции сервера пула путем отправки шаблонов блоков майнинговой аппаратуре. В P2Pool отдельные майнеры строят свои собственные блоки-кандидаты, агрегируют транзакции так же, как соло-майнеры, но потом майнят совместно на ???share chain???. P2Pool представляет собой гибридный подход, который имеет преимущество перед соло-майнингом в виде гораздо более частых выплат, но не давая слишком много контроля оператору пула.

В последнее время количество участников P2Pool значительно возросло в следствие концентрации майнинга в пулах до угрожающих уровней, делающих возможным атаку 51% (см [Атаки на консенсус](#)). Дальнейшее развитие протокола P2Pool продолжается и в будущем возможен отказ от необходимости запуска полного узла и, следовательно, делает децентрализованный майнинг еще проще в использовании.

Даже если P2Pool снижает концентрацию власти операторов майнинговых пулов, атака 51% на саму сеть очевидно остается возможной. Гораздо более широкое принятие P2Pool не решает проблему атаки 51% на сам Bitcoin. Скорее всего, P2Pool делает Bitcoin более устойчивым в целом, в рамках диверсифицированной майнинговой экосистемы.

Атаки на консенсус

Механизм консенсуса в Bitcoin, по крайней мере теоретически, уязвим для атак майнеров (или

пулов), которые могут попытаться использовать их мощности хеширования для нечестных или разрушительных целей. Как мы видели, механизм консенсуса зависит от наличия большинства майнеров, действующих честно из корысти. Однако, если майнер или группа майнеров может заполучить значительной долю майнинговых мощностей, они могут атаковать механизм консенсуса с целью нарушения безопасности и доступности сети Bitcoin.

Важно отметить, что атаки на консенсус могут повлиять только на будущий консенсус, или в лучшем случае, на недавнее прошлое (десятки блоков). Бухгалтерская книга Bitcoin становится все более и более неизменяема с течением времени. В то время как в теории, форк может случиться на любой глубине, на практике, вычислительная мощность, требуемая для возникновения очень глубокого форка настолько огромна, что делает старые блоки практически неизменными. Атаки на консенсус также не влияют на безопасность частных ключей и алгоритм подписи (ECDSA). Атаки на консенсус не могут украсть биткойны, потратить биткойны без подписей, перенаправить биткойны, или иным образом изменить прошлые транзакции или записи собственности. Атаки консенсуса могут влиять только на самые последние блоки и вызвать отказ в обслуживании при создании будущих блоков.

Один из сценариев атаки на механизм консенсуса называется "атакой 51%". В этом случае группа майнеров, контролируя большинство (51%) суммарной мощности хеширования сети, договариваются атаковать Bitcoin. Благодаря возможности добывать большинство блоков, атакующие майнеры могут умышленно вызвать "форки" в blockchain, устроить повторное расходование транзакций или выполнить атаки на отказ в обслуживании против конкретных транзакций или адресов. Форк/повторное расходование — это такие атаки, где злоумышленник может заставить ранее подтвержденные блоки быть признаны недействительными, породив под ними альтернативную цепочку. При наличии достаточных мощностей, злоумышленник может аннулировать шесть или более блоков подряд, в результате чего операции, которые считались неизменными (шесть подтверждений) должны быть признаны недействительными. Обратите внимание, что повторное расходование может быть сделано только на собственных транзакциях атакующего, для которой злоумышленник может предоставить корректную подпись. Повторное расходование имеет смысл, если путем объявления транзакции недействительной, злоумышленник может получить необратимый обменный платеж или получив какой-либо продукт, отменить платеж за него.

Рассмотрим практический пример атаки 51%. В первой главе мы рассмотрели транзакции Алисы Бобу за чашку кофе. Боб, владелец кафе, готов принимать оплату за чашку кофе, не дожидаясь подтверждения (нахождения блока), потому что риск повторного расходования платежа за чашку кофе мал по сравнению с удобством быстрого обслуживания клиентов. Это похоже на то, как в кофейнях, принимают платежи по кредитным картам без подписи для сумм ниже \$25, так как риск возвратного платежа по карте является низким, а стоимость задержки транзакции с требованием подписи сравнительно больше. Наоборот, продажа более дорогого товара за биткойны создает риск атаки повторного расходования, где покупатель распространяет конкурирующую транзакцию, которая расходует те же входы (UTXO) и отменяет платеж продавцу. Повторное расходование может происходить двумя способами: либо атакующий пытается успеть до подтверждения транзакции, либо если злоумышленник использует форк блокчейна для отмены нескольких блоков. Атака 51% позволяет

злоумышленникам повторное расходование своих транзакций в новой цепи, тем самым отменяя соответствующую транзакцию в старой цепи.

В нашем примере, злоумышленник Мэллори отправляется в галерею Кэрол и приобретает красивый триптих, изображающий Сатоси Накамото в образе Прометея. Кэрол продает Мэллори "Великий Огонь" картины за \$250000 в биткоинах. Вместо того чтобы ждать в течение шести или более подтверждений транзакции, Кэрол вручает картину Мэллори только после одного подтверждения. Мэллори работает с сообщником, Полом, который управляет большим майнинг-пулом и тот начинает атаку 51%, как только видит, что транзакция Мэллори включается в блок. Пол направляет пул на повторный майнинг блока на той же высоте, на которой находился блок с транзакцией Мэллори, где заменяет платеж Мэллори транзакцией повторного расходования того же входа, что использовала Мэллори. Транзакция повторного расходования израсходует тот же UTXO и заплатит его обратно в кошелек Мэллори, вместо того, чтобы заплатить Кэрол. Пол затем направит майнинг на поиск дополнительного блока, с тем, чтобы сделать цепь, содержащую транзакцию повторного расходования длиннее исходной цепи (создав форк ниже блока, содержащего транзакцию Мэллори). Когда сеть решит в пользу более длинного нового форка блокчейна, транзакция повторного расходования заменит исходный платеж на адрес Кэрол. Кэрол теперь не хватает трех картин и денег за них она не получит. Пока происходили все описанные события, участники майнинг-пула Пола находились в блаженном неведении о том, как использовались их мощности.

Для защиты от подобного рода атак, продавцы, продающие на крупные суммы, должны ждать по крайней мере шесть подтверждений, прежде чем отдать продукт покупателю. В качестве альтернативы, торговец может использовать условное депонирование в мультиподписной схеме, подождав опять же несколько подтверждений после того, как на счету условного депонирования появятся средства. Чем больше подтверждений пройдет, тем труднее становится отменить транзакцию при помощи атаки на 51%. Для товаров с высокой стоимостью, оплата в Bitcoin будут по-прежнему удобной и эффективной, даже если покупателю придется ждать 24 часа, соответствующих примерно 144 подтверждениям.

В дополнение к атаке на повторное расходование, другой сценарий для атаки на консенсус — это отказ в услуге для конкретных участников сети Биткоин (конкретных Биткоин-адресов). Злоумышленник с большими майнинговыми мощностями может просто игнорировать определенные транзакции. Если они включены в блок, добытый другим майнером, злоумышленник может намеренно форкнуть и перемайнить блок, опять же исключив из него определенные транзакции. Этот тип атаки может привести к устойчивому отказу в обслуживании против конкретного адреса или набора адресов, пока атакующий контролирует большие майнинговые мощности.

Несмотря на свое название, сценарий атаки 51% фактически не требует 51% от мощностей хеширования. На самом деле, такая атака может быть предпринята с меньшим процентом мощностей. Порог 51% просто уровень, при котором такая атака почти гарантированно будет успешной. Атака на консенсус по существу — перетягивание каната за следующий блок, где скорее всего победит более «сильный» участник. С меньшим количеством мощностей хеширования, вероятность успеха снижается, потому что другие майнеры будут

контролировать обнаружение некоторых блоков «по-честному». На это можно посмотреть так, что чем больше мощностей хеширования у злоумышленника, тем больше форков он способен создать намеренно и тем больше блоков из недавнего прошлого он сможет аннулировать, или создание тем большего количества блоков в будущем он может контролировать. Группы исследователей безопасности использовали статистическое моделирование, чтобы доказать, что различные типы атак на консенсус возможны обладая всего лишь 30% от общих мощностей хеширования.

Массивное увеличение общей мощности хеширования сделало Bitcoin невосприимчивым к атакам со стороны соло-майнеров. Подобный одиночка не имеет никакой возможности распоряжаться более чем небольшим процентом от общей мощности майнинга. Однако, централизация контроля, вызванная объединением в майнинг-пулы создала риск для атак со стороны операторов пулов. Оператор управляемого пула контролирует построение блоков-кандидатов, а также контролирует, какие транзакции в них включаются. Это дает оператору пула власть исключать транзакции или проводить транзакции повторных расходов. Если такое злоупотребление властью происходит в ограниченном объеме, то оператор пула вполне может получить прибыль от атаки на консенсус, не будучи замеченным.

Однако, не все злоумышленники будут мотивированы прибылью. Один из возможных сценариев атаки, это когда злоумышленник намерен сорвать работу сети Bitcoin без возможности заработать на этом. Вредоносная атака направленная на нарушение работы Bitcoin потребует огромных инвестиций и тайное планирование, но вполне может быть запущена хорошо финансируемым, скорее всего, спонсируемым государством, злоумышленником. С другой стороны, хорошо финансируемый злоумышленник может атаковать консенсус Bitcoin путем одновременного накопления оборудования для майнинга, компрометации операторов пулов и устроив атаки на отказ обслуживания других пулов. Все эти сценарии теоретически возможны, но со временем все менее практичны по причине экспоненциального роста мощностей хэширования сети Bitcoin.

Несомненно, серьезная атака на консенсус подорвет доверие к Bitcoin в краткосрочной перспективе, вызвав значительное снижение цены. Тем не менее, сеть Bitcoin и программное обеспечение в ее основе, постоянно развиваются, так что атаки на консенсус будут встречены контрмерами со стороны сообщества.

Альтернативные цепи, валюты, `<phrase role="keep-together">`и приложения`</phrase>`

Биткоин явился результатом 20-ти лет исследований в распределенных системах и валютах и принес новую революционную технологию: децентрализованный механизм консенсуса на основе доказательства работы. Это изобретение в центре Биткоин открыл волну инноваций в деньгах, финансовых услугах, экономике, распределенных системах, системах голосования, корпоративного управления, и договорах.

В этой главе мы рассмотрим многие ответвления Биткоин и blockchain: альтернативные цепочки, валюты, и приложения, построенные с внедрением этой технологии после появления ее в 2009 году. Главным образом, мы рассмотрим *альтернативные валюты*, или *alt coins*, которые представляют собой цифровые валюты, построенные по принципам Биткоин, но со своими блокчейнами и сетью.

На один альткоин, упомянутый в этой главе приходится 50 или больше не упомянутых. Целью этой главы является не оценка альткоинов или даже просто перечисление наиболее значимых из них на основании некоторой субъективной оценки. Вместо этого, мы выделим несколько примеров, которые показывают широту и разнообразие экосистемы, отмечая инновационные или важные различия. Некоторые из наиболее интересных примеров альткоинов на самом деле являют полную неудачу в финансовом плане. Это, возможно, делает их еще более интересными для изучения и подчеркивает тот факт, что эта глава не должна использоваться в качестве справочника инвестора.

Так как каждый день появляется все больше новых альткоинов, очень легко пропустить какой-нибудь из них, который изменит историю. Скорость инноваций просто захватывающая и гарантирует, что эта глава будет неполной и устаревшей сразу после публикации.

Таксономия альтернативных валют и цепей

Биткоин — это проект с открытым исходным кодом, и этот код использовался в качестве основы для многих других программных проектов. Чаще всего исходный код Биткоин используется для создания альтернативных децентрализованных валют, или *альткоинов*.

Поверх блокчейна Биткоина реализован ряд дополнительных слоев. Эти *матамонеты*, *метацепочки* или *приложения блокчейна* используют блокчейн в качестве платформы приложений или расширяют протокол Биткоин, добавляя свои протоколы. Примеры включают в себя Colored Coins, Mastercoin, NXT, и Counterparty.

В следующем разделе мы рассмотрим несколько известных альткоинов, таких как Litecoin, Dogecoin, Freicoin, Primecoin, Peercoin, Darkcoin и Zerocoin. Эти альткоины знамениты по

историческим причинам или потому, что явились хорошими примерами конкретных инноваций, а не из-за своей капитализации.

В дополнение к альткоином, есть также ряд альтернативных реализаций блокчейна, которые на самом деле не "монеты" и которые я называю *альтернативными цепями*. Эти альтернативные цепи реализуют алгоритм консенсуса и распределенной реестр в качестве платформы для контрактов, регистрации доменов, или других приложений. Альтернативные цепи используют те же самые основные строительные блоки, а иногда и используют валюту в качестве механизма оплаты, но их основная цель — не деньги. Мы рассмотрим Namecoin и Ethereum в качестве примеров альтернативных цепей.

Наконец, есть ряд конкурентов Биткоина, в смысле цифровой валюты или платежной сети, но без использования децентрализованного реестра или механизма консенсуса на основе доказательства работы, такие, как Ripple и другие. Эти не основанные на блокчейне технологии выходят за рамки этой книги, а не будут рассмотрены в этой главе.

Платформы метакринов

Метакрины и метаципи — это дополнительные слои ПО, реализованного поверх Биткоин, либо в виде валюты-внутри-валюты, или в виде платформы/протокола внутри системы Биткоин. Эти функциональные слои расширяют протокол ядра Биткоин и добавляют новые функции и возможности путем кодирования дополнительных данных внутри транзакций Биткоин и Биткоин-адресов. Первые реализации метакринов использовали различные хаки, чтобы добавить метаданные в блокчейн, вроде использования Биткоин-адресов для кодирования данных или с помощью неиспользуемых полей транзакций (например, поля последовательности в транзакциях) для кодирования метаданных о добавленном слое протокола. С момента введения опкода OP_RETURN в скрипты транзакций, метакрины уже были в состоянии записывать метаданные в блокчейн более прямым методом, и большинство из них мигрируют в этом направлении.

Colored Coins

Colored coins — это метапротокол, который накладывает информацию поверх небольших Биткоин-транзакций. "Цветная" монета — это некоторое количество биткоинов, выражающих совершенно другой актив. Представьте себе, например, что кто-то берет купюру стоимостью в \$1 и ставит на ней штамп, говоря, "Теперь это сертификат 1 доли Acme Inc." Теперь эта денежная купюра служит двум целям: это банкнота, но это также сертификат доли. Так как в виде ценной бумаги купюра более ценна, вы не станете ее использовать для покупки конфет. Поэтому де факто — это уже не валюта. Цветные монеты работают таким же образом, путем преобразования конкретного, очень небольшого количества биткоинов в обращающийся сертификат, который представляет собой другой актив. Термин "цвет" относится к идее выделения особенности, добавления атрибута, такого как цвет. В данном случае цвет — это метафора, а не ассоциация с фактическим цветом.

Цветные монеты управляются специализированными кошельками, которые записывают и

анализируют метаданные, прикрепленные к цветным биткоином. Используя подобный кошелек, пользователь может конвертировать сумму биткоинов из неокрашенной валюты в цветные монеты, добавляя метку, которая имеет особое назначение. Например, метка может представлять сертификаты акций, купоны, недвижимость, товары или коллекционные жетоны. От пользователей цветных монет полностью зависит то, какое значение они придают и как интерпретируют смысл "цвета", связанный с конкретными монетами. Для того, чтобы придать монете цвет, пользователь определяет соответствующие метаданные, такие как тип эмиссии, возможность дробления на более мелкие единицы, символ и описание, а также другую информацию. После того, как монеты окрашены, они могут быть куплены и проданы, дробиться и складываться, а также получать выплаты дивидендов. Цветные монеты могут также быть "обесцвечены", путем удаления специальной ассоциации и выкупа их номинальной стоимости в биткоинах.

Для демонстрации использования цветных монет, мы создали набор из 20 цветных монет с символом "MasterBTC", которые представляют собой талоны на бесплатную копию этой книги, показанных на [Профиль метаданных цветных монет записанный в виде купона на бесплатную копию книги](#). Каждая единица MasterBTC, в виде этих цветных монет, теперь может быть продана или передана любому Bitcoin пользователю с кошельком, поддерживающим цветные монеты, который затем сможет передать их другим пользователям или обменять их у эмитента на бесплатную копию книги. Этот пример цветных монет можно увидеть [here](#).

Example 1. Профиль метаданных цветных монет записанный в виде купона на бесплатную копию книги

```
{
  "source_addresses": [
    "3NpZmvSPLmN2cVfw1pY7gxEAVPCVfnWfVD"
  ],
  "contract_url":
  "https://www.coinprism.info/asset/3NpZmvSPLmN2cVfw1pY7gxEAVPCVfnWfVD",
  "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 \"Mastering 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"
}
```

Mastercoin

Mastercoin представляет собой слой протоколов поверх Bitcoin, который поддерживает платформу для различных приложений, расширяющих систему Bitcoin. В Mastercoin для транзакций используется валюта MST, но это скорее не валюта, а платформа для создания других вещей, таких как пользовательские валюты или децентрализованные обмены активами и контрактами. Mastercoin можно представить в качестве протокола прикладного уровня на вершине транспортного уровня транзакций Bitcoin, по аналогии того, как HTTP работает поверх TCP.

Mastercoin работает в основном за счет транзакций, посланных в/из специального Bitcoin-адреса, называемого адресом "исхода" (1EXoDusjGwvnjZUyKkxZ4UHEf77z6A5S4P), так же, как для HTTP используется определенный TCP-порт (порт 80), чтобы отличать его трафик от остальной части TCP трафика. Протокол Mastercoin постепенно переходит от использования специализированного адреса исхода и мульти-подписи на использование оператора OP_RETURN для кодирования метаданных транзакций.

Counterparty

Counterparty—это еще один слой протокола, реализованный поверх Bitcoin. Counterparty позволяет валюты пользователей, торгуемые жетоны, финансовые инструменты, децентрализованный обмен активами и другие функции. Counterparty осуществляется в первую очередь с помощью оператора OP_RETURN языка сценариев Bitcoin для записи метаданных, что придает Bitcoin-транзакциям дополнительный смысл. В качестве единицы обмена в транзакциях Counterparty используется валюта XCP.

Альткоины

Подавляющее большинство альткоинов происходят от исходного кода Bitcoin, поэтому их называют "форками" ("fork"—вилка, ответвление, англ, прим. перев.). Некоторые из них реализованы "с нуля" на основе модели blockchain, но без использования какой-либо из исходного кода Bitcoin. Альткоины и альтернативные цепочки (в следующем разделе) оба предлагают отдельные реализации технологии blockchain и обе формы используют свои собственные blockchain. Такое различие терминов существует, чтобы указать, что альткоины в основном используются в качестве валюты, в то время как альтернативные цепочки используются для других целей, не в первую очередь в качестве валюты.

Строго говоря, первым крупным "альтернативным" форком кода Биткойн был не альткойн, а альтернативная цепь *Namescoin*, которую мы будем обсуждать в следующем разделе.

Основываясь на дату объявления, первый альткойн, который была ответвлен от Биткойн появился в августе 2011 года; он назывался *IXCoin*. В *IXCoin* изменены некоторые параметры Биткойн, в частности, ускорено создание валюты за счет увеличения вознаграждение до 96 монет за блок.

В сентябре 2011 года был запущен *Tenebrix*. Tenebrix был первой криптовалютой, реализующей альтернативный алгоритм доказательства работы, а именно *scrypt*, алгоритм изначально предназначенный для растягивания паролей (для противодействия перебору). В качестве целей Tenebrix заявлялась устойчивость к майнингу на GPU и ASIC, с помощью алгоритма, интенсивно использующего память. Tenebrix не увенчался успехом в качестве валюты, но это стал основой для Litecoin, который пользуется большим успехом и породил сотни клонов.

Помимо использования Scrypt в качестве алгоритма доказательства работы, время генерации нового блока в *Litecoin* также сокращено до 2,5 минут вместо 10 минут в Bitcoin. Данная валюта позиционируется как "серебро относительно золотого Bitcoin» и задумана в качестве легковесной альтернативной валюты. Из-за более быстрого времени подтверждения блока и общего предела валюты в 84 миллиона, многие приверженцы Litecoin считают, что она лучше, чем Bitcoin подходит для розничных транзакций.

Альткоины на основе Bitcoin и Litecoin продолжили распространение в 2011 и 2012 годах. К началу 2013-го года насчитывалось 20 альткоинов, борющихся за позиции на рынке, но уже к концу года эта цифра резко увеличилась до 200 так, что 2013 можно назвать "годом альткоинов". Рост продолжился в 2014 году и количество альткоинов достигло числа 500 на момент написания этих строк. Более половины альткоинов сегодня являются клонами Litecoin.

Создание альткоинов просто, поэтому-то в настоящее время их существует более 500. Большинство альткоинов очень незначительно отличаются от Bitcoin и не предлагают ничего стоящего. Многие из них на самом деле созданы просто в качестве попытки обогатить своих создателей. Среди подражателей и схем "накачка и сброс", есть, однако, некоторые заметные исключения и очень важные нововведения. Эти альткоины принимают совершенно разные подходы или добавляют значительные инновации в дизайн Bitcoin. Эти альткоины отличаются от Bitcoin в трех основных моментах:

- Различная монетарная политика
- Различные механизмы доказательства работы или консенсуса
- Особенности, такие как анонимность

Для получения дополнительной информации посетите этот сайт [graphical timeline of alt coins and alt chains](#).

Оценка альткоинов

Имея такое большое количество альткоинов, как же решить, какие из них достойны внимания? Некоторые альткоины пытаются достичь широкого распространения и использования в качестве валюты. Другие представляют собой лаборатории для экспериментов над различными функциями и монетарными моделями. Многие из них представляют собой просто схемы быстрого обогащения их создателей. Для оценки альткоинов, я смотрю на их отличительные особенности и их рыночные показатели.

Вот некоторые вопросы, которыми стоит задаться в поисках отличий альткоина от Биткоина:

- Привнес ли альткойн какую-либо значительную инновацию?
- Является ли отличие достаточно убедительным, чтобы привлечь пользователей Биткойн?
- Обращен ли альткойн к какой-либо интересной рыночной нише?
- Может альткойн привлечь достаточное количество майнеров для защиты от атак на консенсус?

Вот для рассмотрения некоторые ключевые финансовые и рыночные показатели:

- Какова общая рыночная капитализация альткойна?
- Каковы оценки количества пользователей/кошельков альткойна?
- Сколько продавцов принимают альткойны?
- Сколько у альткойна транзакций в день?
- Сколько ежедневно транзакций в денежном эквиваленте?

В этой главе мы сосредоточимся в основном на технических характеристиках и инновационном потенциале альткойнов.

Альтернативные монетарные параметры: Litecoin, Dogecoin, Freicoin

Bitcoin имеет несколько денежных параметров, которые дают ему отличительные характеристики дефляционной валюты с фиксированной эмиссией. Эмиссия ограничена до 21 млн основных валютных единиц (или 21 квадриллионов мелких единиц), скорость снижения эмиссии имеет вид геометрической прогрессии, и у него есть 10-минутное "сердцебиение", которое контролирует скорость подтверждения транзакций и эмиссии валюты. Многие альткойны изменили основные параметры для достижения различных монетарных политик. Некоторые из сотен примеров альткойнов, включают в себя.

Litecoin

Один из первых альткойнов, выпущенный в 2011 году. Litecoin — это вторая по успешности цифровая валюта после Биткойн. Его основные инновациями были использование *script* в качестве алгоритма доказательства работы (унаследованный из Tenebrix) и более скоростные/легкие параметры валюты.

- Время генерации блока: 2.5 минуты
- Всего монет: 84 млн монет до 2140 года
- Алгоритм консенсуса: доказательство работы на основе Scrypt
- Рыночная капитализация: \$160 млн в середине-2014 года

Dogecoin

Dogecoin был запущен в декабре 2013 года, на основе форка Litecoin. Dogecoin примечателен своей быстрой эмиссией, которая стимулирует пользователей расходовать валюты, а также

жертвовать чаевые. Dogecoin был основан как шутка, но стал весьма популярным, заработал большое и активное сообщество, прежде, чем быстро угаснуть в 2014 году.

- Время генерации блока: 60 секунд
- Всего монет: 100,000,000,000 (100 миллиардов) до 2015 года
- Алгоритм консенсуса: доказательство работы на основе Scrypt
- Рыночная капитализация: \$12 млн. в середине 2014 года

Freicoin

Freicoin был запущен в июле 2012 года это *демереджная валюта*, т.е. на хранимое значение действует отрицательная процентная ставка. Стоимость хранимая в Freicoin облагается комиссионными объемом 4,5% годовых для стимулирования потребления и препятствования накопительству. Freicoin отличается тем, что она реализует денежно-кредитную политику, которая является точной противоположностью дефляционной политики Bitcoin. Freicoin не была успешна в качестве валюты, но это интересный пример различных денежно-кредитных политик, которые могут быть выражены с различных в альткоинах.

- Время генерации блока: 10 минут
- Всего монет: 100 млн монет по 2140 года
- Алгоритм консенсуса: доказательство работы на основе SHA256
- Рыночная капитализация: \$130,000 в середине 2014 года

Инновации консенсуса: Peercoin, Myriad, Blackcoin, Vericoin, NXT

Механизм консенсуса Bitcoin основан на доказательстве работы с использованием алгоритма SHA256. Первые альткоины использовали Scrypt в качестве альтернативного алгоритма доказательства работы, что сделать майнинг более удобным на обычных процессорах и менее восприимчивы к централизации из-за применения ASIC. С тех пор инновации в механизме консенсуса продолжались в лихорадочном темпе. Несколько альткоинов приняли целый ряд алгоритмов, подобных Scrypt, scrypt-N, Skein, Groestl, SHA3, X11, Blake, и другие. Некоторые альткоины объединяли несколько алгоритмов для доказательства работы. В 2013 году мы увидели изобретение альтернативы доказательству работы под названием *proof of stake*, который лежит в основе многих современных альткоинов.

Proof of stake — это система, в которой существующие владельцы валюты могут зарезервировать часть своей валюты в качестве "доли", зарабатывая при этом инвестиционную отдачу от новой валюты (выдается в качестве процентных платежей) и комиссионных от транзакций.

Peercoin

Peercoin была запущен в августе 2012 года и стал первым альткоином, использовавшим гибридный алгоритм proof-of-work и proof-of-stake.

- Время генерации блока: 10 минут
- Всего монет: не ограничено
- Алгоритм консенсуса: (гибридный) proof-of-stake после доказательства работы
- Рыночная капитализация: \$14 млн в середине-2014 года

Myriad

Myriad была запущена в феврале 2014 года и отличается одновременным использованием пяти различных алгоритмов доказательства работы (SHA256d, Scrypt, Qubit, Skein, или Myriad-Groestl), с варьированием сложности каждого алгоритма в зависимости от участия майнеров. Задача состоит в том, чтобы придать Myriad иммунитет к ASIC, а также улучшить сопротивляемость атакам на консенсус, поскольку несколько майнинговых алгоритмов должны быть атакованы одновременно.

- Время генерации блока: в среднем каждые 30 секунд (2.5 минут для каждого майнингового алгоритма)
- Всего монет: 2 млрд до 2024 года
- Алгоритм консенсуса: комплексное доказательство работы
- Рыночная капитализация: \$120,000 в середине-2014 года

Blackcoin

Blackcoin был запущен в феврале 2014-го года и использует алгоритм консенсуса proof-of-stake. Для этой монеты впервые появилось понятие мультипула ("multipools"), специального майнингово пула, который может переключаться между различными альткоинами автоматически, в зависимости от рентабельности.

- Время генерации блока: 1 минута
- Всего монет: не ограничено
- Алгоритм консенсуса: Proof-of-stake
- Рыночная капитализация: \$3.7 млн в середине-2014 года

VeriCoin

VeriCoin был запущен в мае 2014 года. В нем используется proof-of-stake с переменной процентной ставкой, которая динамически регулируется на основе рыночных сил спроса и предложения. Он также является первым альткоином с автоматическим обменом на Bitcoin прямо из кошелька.

- Время генерации блока: 1 минута
- Всего монет: не ограничено
- Алгоритм консенсуса: Proof-of-stake

- Рыночная капитализация: \$1.1 млн в середине-2014 года

NXT

NXT (произносится как "Next") — это альткойн на основе "чистого" proof-of-stake в том смысле, что в нем не используется майнинг для доказательства работы. NXT был написан с нуля и не является форком Bitcoin или любого другого альткойна. NXT реализует множество дополнительных функций, в том числе реестр имен (по аналогии с Namecoin), децентрализованный обмен активами (по аналогии с цветными монетами), интегрированный децентрализованный и безопасный обмен сообщениями (похожий на Bitmessage), и делегирование долей (для делегирования proof-of-stake другим). Приверженцы NXT называют ее криптовалютой 2.0 или "следующим поколением".

- Время генерации блока: 1 минута
- Всего монет: не ограничено
- Алгоритм консенсуса: Proof-of-stake
- Рыночная капитализация: \$30 млн в середине-2014 года

Инновация в майнинге двойного назначения: Primecoin, Curecoin, Gridcoin

Алгоритм доказательства работы Bitcoin имеет только одну цель: обеспечение безопасности сети Bitcoin. По сравнению с системами безопасности традиционных платежных систем, стоимость майнинга не очень высока. Тем не менее, она подвергается критике за "бесполезные расходы". Следующее поколение альткойнов пытаются решить эту проблему. Алгоритмы доказательства работы двойного назначения служат для решения конкретной «полезной» задачи, попутно производя доказательство работы для обеспечения сети. Риск добавления внешнего использования для безопасности валюты является в том, что он также добавляет внешнее влияние на кривую спроса/предложения.

Primecoin

Primecoin был запущен в июле 2013 года. Его алгоритм доказательства работы ищет особые последовательности простых чисел особого вида — последовательности Куннингама. Простые числа полезны в различных научных дисциплинах. Блокчейн Primecoin не просто обслуживает денежные переводы, но и содержит в себе обнаруженные простые числа, в результате чего результаты становятся публично доступны в научных целях.

- Время генерации блока: 1 минута
- Всего монет: не ограничено
- Алгоритм консенсуса: доказательство работы с поиском последовательностей простых чисел
- Рыночная капитализация: \$1.3 млн в середине 2014 года

Curecoin

Curecoin был запущен в мае 2013 г. Этот альткойн совмещает в себе алгоритм доказательства работы на основе SHA256 и моделирование процессов свертывания молекул белка для проекта Folding@Home. Моделирование свертывания белка помогает в поиске новых лекарств и причин возникновения некоторых болезней.

- Время генерации блока: 10 минут
- Всего монет: не ограничено
- Алгоритм консенсуса: доказательство работы с исследованием фолдинга белков
- Рыночная капитализация: \$58,000 в середине-2014 года

Gridcoin

Gridcoin была запущена в октябре 2013 г. Помимо доказательства работы на основе алгоритма Scrypt, в ней майнеры участвуют в системе грид-вычислений BOINC. BOINC (Berkeley Open Infrastructure for Network) — это открытый протокол для проведения научных исследований на основе распределенных вычислений, участники которого жертвуют циклы простоя своих процессоров для широкого спектра академических научных вычислений. Gridcoin использует BOINC в качестве источника заданий общего назначения, а не определенных проблем вроде поиска простых чисел или способов сворачивания белка.

- Время генерации блока: 150 секунд
- Всего монет: не ограничено
- Алгоритм консенсуса: доказательство работы с использованием распределенных вычислений BOINC
- Рыночная капитализация: \$122,000 в середине-2014 года

Альткойны с фокусированием на анонимность: CryptoNote, Bytecoin, Monero, Zerocash/Zerocoin, Darkcoin

Bitcoin часто ошибочно характеризуется, как "анонимная" валюта. На самом деле, относительно легко связать пользователей с Биткойн-адресами и, используя аналитику больших данных, связать адреса друг с другом и получить полную картину чьих-то покупательских привычек. Несколько альт-койнов направлены на решение этой проблемы непосредственно, сосредоточив внимание на сильной анонимности. Первая такая попытка, скорее всего *Zerocoin*, протокол мета-монеты для сохранения анонимности поверх Bitcoin, впервые опубликованный в 2013. Zerocoin будет реализован в виде полностью отдельного альт-койна под названием Zerocash, находящегося в процессе разработки на момент написания книги. Альтернативный подход к анонимности, был обозначен для *CryptoNote* в статье, опубликованной в октябре 2013 г. CryptoNote находится в основе некоторых других форков, обсуждающихся далее. В дополнение к Zerocash и CryptoNotes, есть несколько других независимых анонимных монет, таких как Darkcoin, которые используют стелс-адреса или

транзакции повторного смешивания для анонимности.

Zerocoin/Zerocash

Zerocoin — это теоретический подход к анонимной цифровой валюте, опубликованный в 2013 году исследователями из университета Джона Хопкинса. Zerocash — это альткойн, реализующий идеи Zerocoin, находится в разработке и пока не выпущен.

CryptoNote

CryptoNote — это эталонная реализация альткойна, служащего основой для анонимных цифровых наличных. Проект был запущен в октябре 2013 г. Его основное предназначение быть эталонной реализацией для различных других проектов и в его коде даже присутствует встроенный механизм периодического сброса, который делает сам CryptoNote непригодным для использования в качестве валюты. От CryptoNote было отпочковано сразу несколько альткойнов, включая Bytecoin (BCN), Aeon (AEON), Boolberry (BBR), duckNote (DUCK), Fantomcoin (FCN), Monero (XMR), MonetaVerde (MCN), и Quazarcoin (QCN). CryptoNote также примечателен тем, что не является форком Биткойна, а написан с самого начала с нуля.

Bytecoin

Bytecoin был запущен в июле 2012 года и представлял собой жизнеспособную анонимную валюту на основе технологии CryptoNote. До этого уже существовал альткойн с названием Bytecoin и символом BTE, в то время как Bytecoin на основе CryptoNote имеет обозначение BCN. Bytecoin использует алгоритм доказательства работы Cryptonight, который требует доступ к по меньшей мере, 2 МБ оперативной памяти, что делает его непригодным для GPU или ASIC-майнинга. Bytecoin наследует кольцевую подпись, несвязываемые транзакции и анализоустойчивую анонимность из CryptoNote.

- Время генерации блока: 2 минуты
- Всего монет: 184 млрд BCN
- Алгоритм консенсуса: доказательство работы Cryptonight
- Рыночная капитализация: \$3 млн в середине-2014 года

Monero

Monero — это еще одна реализация на основе CryptoNote. В ней присутствует более пологая, чем в Bytecoin кривая эмиссии, с выпуском 80% монет в течение первых четырех лет. Она предлагает все возможности анонимности, унаследованные от CryptoNote.

- Время генерации блока: 1 минута
- Всего монет: 18.4 млн XMR
- Алгоритм консенсуса: доказательство работы Cryptonight
- Рыночная капитализация: \$5 млн в середине-2014 года

Darkcoin

Darkcoin был запущен в январе 2014 г. Darkcoin представляет собой анонимную валюту с использованием протокола перемешивания транзакций, называемого DarkSend. Darkcoin также отличается использованием последовательности из 11 различных хэш-функций (blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, echo) в алгоритме доказательства работы.

- Время генерации блока: 2.5 минуты
- Всего монет: максимум 22 миллиона DRK
- Алгоритм консенсуса: многоалгоритмовое многоаундовое доказательство работы
- Рыночная капитализация: \$19 миллионов в середине-2014

Альтернативные цепочки без валюты

Альтернативные цепочки или альтчейны — это такие реализации цепочек блоков, первичное предназначение которых не для использования в качестве валюты. Многие из них включают в себя валюту, но валюта здесь используется в качестве маркера для выделения чего-то еще, например, ресурса или контракта. Валюта, другими словами — это не точка опоры платформы, а ее вторичный признак.

Namescoin

Namescoin был первым ответвлением кода Bitcoin. Namescoin представляет собой децентрализованную платформу ключ-значение на основе blockchain. Он поддерживает глобальный реестр доменных имен, похожий на систему регистрации доменных имен в Интернете. Namescoin в настоящее время используется в качестве альтернативы службе доменных имен (DNS) для домена корневого уровня .bit. Namescoin также может использоваться для регистрации имен и пар ключ-значение в других пространствах имен; для хранения данных вроде адресов электронной почты, ключей шифрования, SSL сертификатов, подписей файлов, систем голосования, сертификатов акций; и во множестве других приложений.

Система Namescoin включает в себя валюту Namescoin (символ NMC), которая используется для оплаты комиссионных транзакций регистрации и трансфера имен. В текущих ценах стоимость регистрации имени составляет 0.01 NMC или примерно 1 цент США. Как и в Bitcoin, сборы взимаются namescoin-майнерами.

Основные параметры Namescoin такие же, как у Биткойн:

- Время генерации блока: 10 минут
- Всего монет: 21 млн NMC до 2140 года
- Алгоритм консенсуса: доказательство работы на основе SHA256
- Рыночная капитализация: \$10 млн в середине-2014 года

Пространства имен в Namecoin не ограничены и могут использоваться как угодно. Тем не менее, некоторые пространства имен имеют согласованные спецификации, так что программное обеспечение прикладного уровня должно понимать их предназначение и формат. Если формат неправильный, то программное обеспечение должно сообщить об ошибке. Вот некоторые из популярных пространств имен:

- `d/` — это пространство имен для доменов `.bit`
- `id/` — это пространство имен для хранения человеческих идентификаторов, таких, как адреса электронной почты, ключи PGP, и так далее
- `u/` — это дополнительная, более структурированная спецификация для хранения идентичностей (на основе openspecs)

Клиент Namecoin очень похож на Bitcoin Core, поскольку он является производным от того же исходного кода. После установки, клиент должен загрузить полную копию блокчейна Namecoin, а затем будет готов к запросам и регистрации имен. Есть три основные команды:

name_new

запрос или предварительная регистрация имени

name_firstupdate

Регистрация и публикация имени

name_update

Изменение деталей или обновление регистрации имени

Например, чтобы зарегистрировать домен `mastering-bitcoin.bit`, используем команду `name_new` следующим образом:

```
$ namecoind name_new d/mastering-bitcoin
```

```
[  
  "21cbab5b1241c6d1a6ad70a2416b3124eb883ac38e423e5ff591d1968eb6664a",  
  "a05555e0fc56c023"  
]
```

Команда `name_new` регистрирует требование на имя, путем создания хэша имени со случайным ключом. Команда возвращает две строки: хэш и случайный ключ (`a05555e0fc56c023` в предыдущем примере), которые могут быть использованы для опубликования регистрации. Как только требование записано в блокчейн Namecoin, оно может быть сконвертировано в публичную регистрацию командой `name_firstupdate`, путем предоставления случайного ключа:

```
$ namecoind name_firstupdate d/mastering-bitcoin a05555e0fc56c023 "{\"map\": {\"www\": {\"ip\": \"1.2.3.4\"}}}}\"  
b7a2e59c0a26e5e2664948946ebca1260985c2f616ba579e6bc7f35ec234b01
```

В этом примере доменное имя `www.mastering-bitcoin.bit` будет отображено в IP-адрес `1.2.3.4`. Возвращенный хеш является идентификатором транзакции и может быть использован для отслеживания этой регистрации. Для того, чтобы посмотреть какие имена зарегистрированы на вас при помощи команды `name_list`:

```
$ namecoind name_list
```

```
[  
  {  
    "name" : "d/mastering-bitcoin",  
    "value" : "{\"map: {www: {ip:1.2.3.4}}}}",  
    "address" : "NCccBXrRUahAGrisBA1BLPWQfSrups8Geh",  
    "expires_in" : 35929  
  }  
]
```

Регистрации в Namecoin необходимо обновлять каждые 36000 блоков (примерно от 200 до 250 дней). Команда `name_update` не требует комиссионных и поэтому обновление доменов в Namecoin является бесплатным. Сторонние поставщики могут обрабатывать регистрацию, автоматическое обновление, и обновление через веб-интерфейс за небольшую плату. Использование стороннего поставщика позволит вам избежать необходимости запускать клиент Namecoin, но вы теряете независимое управление децентрализованным реестром имен предлагаемого Namecoin.

Ethereum

Ethereum — это Тьюринг-полная платформа процессинга и выполнения контрактов на основе блокчейна. Это не клон Bitcoin, но полностью самостоятельный проект. Ethereum имеет встроенную валюту, называемую *эфиром*, которая необходима для оплаты за исполнение контракта. Блокчейн Ethereum содержит *контракты*, на Тьюринг-полном языке низкого уровня по типу байт-кода. По сути, контракт представляет собой программу, которая работает на каждом узле в системе Ethereum. Контракты Ethereum могут хранить данные, отправлять и получать простые платежи в эфирах, хранить эфиры, и выполнять бесконечный диапазон (Тьюринг-полных) вычислимых действий, действуя в качестве децентрализованных автономных программных агентов.

В Ethereum реализуются достаточно сложные системы, которые иначе реализуются в виде

альтернативных цепочек. Например, следующий а-ля Namecoin контракт регистрации написан в Ethereum (точнее, написан на языке высокого уровня, который компилируется в код Ethereum):

```
if !contract.storage[msg.data[0]]: # Ключ уже занят?
    # Тогда взять его!
    contract.storage[msg.data[0]] = msg.data[1]
    return(1)
else:

    return(0) // Иначе ничего не делать
```

Будущее валют

Будущее криптографических валют в целом даже ярче, чем будущее Bitcoin. Bitcoin представил совершенно новую форму децентрализованной организации и консенсуса, который породил сотни невероятных новшеств. Эти изобретения, вероятно, влияют на широкие секторы экономики, финансов, валюты, банков и корпоративного управления. Многие виды человеческой деятельности, которые ранее требовали централизованных институтов или организаций с властными функциями теперь могут быть децентрализованы. Изобретение blockchain и системы консенсуса позволит значительно сократить расходы на организацию и координацию на крупномасштабных системах, при удалении возможности для концентрации власти, коррупции и регуляции.

Безопасность Биткоин

Обеспечение безопасности Биткоин — это сложная задача так, как криптовалюта похожа на наличные или золото, только в цифровом виде. Владение ключами эквивалентно владению пачкой наличных или куском драгоценного металла. Его можно обронить, потерять, он может быть украден, или вы можете случайно выдать кому-то неправильную сумму. В каждом из этих случаев, пользователи не могут рассчитывать на чью-либо помощь.

Тем не менее, у Биткоин есть возможности, которых деньги, золото и обычные электронные деньги не имеют. С Биткоин-кошелек, содержащего ключи, можно делать резервные копии, как с любого файла. Он может храниться в нескольких экземплярах, или даже распечатан на бумаге для надежности. Вы не можете обеспечить "резервное копирование" наличных денег, золота или банковского счета. Биткоин достаточно отличается от всего, что было раньше, поэтому и подходы к безопасности должны быть беспрецедентными.

Принципы безопасности

Главный принцип Биткоин — это децентрализация и этот принцип имеет важные последствия для безопасности. Централизованная модель, такая как, например, традиционный банк или платежная система, сдерживает нечестных игроков основываясь на контроле доступа. Для сравнения, децентрализованная система такая, как Биткоин передает ответственность в руки самих пользователей. Поскольку безопасность сети основана на доказательстве работы, а не контроле доступа, сеть может быть открытой и шифрование Биткоин-трафика не требуется.

В традиционных платежных системах, вроде систем кредитных карт, должно обеспечиваться шифрование, которое бы гарантировало что никакие перехватчики или посредники могли бы поставить под угрозу платежный трафик, по пути передачи или при его сохранении. Если злоумышленник получает доступ к системе, он может поставить под угрозу текущие операции *И* получить доступ к платежные маркеры, которые могут быть использованы для создания новых транзакций. Хуже того, когда данные клиента скомпрометированы, клиенты должны определить пропажу и принять меры для предотвращения незаконного использования взломанных счетов.

Bitcoin резко отличается. Транзакция в Биткоин авторизует лишь конкретное значение для конкретного получателя и не может быть подделана или изменена. Она не раскрывает какой-либо личной информации, такую как имена сторон, и не может быть использована для дополнительных платежей. Таким образом, следует, что платежная сеть Биткоин не должна быть зашифрована или защищена от перехвата. На самом деле, вы можете транслировать Bitcoin транзакции по открытому общественному канала, как например WiFi или Bluetooth, без ущерба безопасности.

Децентрализованная модель безопасности Bitcoin дает много возможностей в руки пользователей. С этими возможностями приходит ответственность за сохранение секретности ключей. Для большинства пользователей это нелегкая задача, особенно на обычных устройствах вроде подключенных к Интернету смартфонов или ноутбуков. Хотя

децентрализованная модель Bitcoin предотвращает массовые утечки, которые время от времени случаются с кредитными картами, многие пользователи не в состоянии должным образом защитить свои ключи и бывают взломаны, один за другим.

Разработка безопасных Bitcoin систем

Самый важный принцип для Bitcoin разработчиков — это децентрализация. Большинство разработчиков знакомы с централизованной моделью безопасности и у них может возникнуть соблазн применить эти модели в своих Bitcoin-приложениях с катастрофическими результатами.

Безопасность Bitcoin опирается на децентрализованное управление над ключами и на независимой проверке сделок майнерами. Если вы хотите положиться на безопасность Bitcoin, то вам следует убедиться, что вы остаетесь в рамках модели безопасности Bitcoin. Проще говоря: не забирайте контроль над ключами у пользователей и предоставьте транзакции блокчейну.

Например, многие ранние Bitcoin-биржи сосредотачивали средства всех пользователей в одном "горячем" кошельке с ключами, хранящимися на одном сервере. Такая конструкция устраняет контроль со стороны пользователей и централизует управление ключами в одной системе. Многие такие системы были взломаны, с катастрофическими последствиями для своих клиентов.

Еще одна распространенная ошибка — это так называемые транзакции "вне блокчейна" ("off blockchain"), которые используются в попытке уменьшить комиссионные издержки или для ускорения обработки транзакций. Система "вне блокчейна" будет записывать транзакции во внутреннем, централизованном реестре и только изредка синхронизировать их с блокчейном Bitcoin. Эта практика, опять-таки, заменяет децентрализованную безопасность Bitcoin на проприетарный централизованный подход. Централизованные реестры могут быть взломаны, а транзакции в них сфальсифицированы.

Если вы не готовы вкладывать значительные средства в оперативную безопасность, несколько слоев контроля доступа и аудиты (как это делают традиционные банки), вы должны очень хорошо подумать, прежде чем выносить средства за пределы контекста децентрализованной безопасности Bitcoin. Даже если у вас есть средства и возможности для реализации надежной модели безопасности, такая конструкция просто повторяет хрупкую модель традиционных финансовых сетей, страдающих от кражи личных данных, коррупции и растрат. Для того, чтобы воспользоваться уникальной децентрализованной моделью безопасности Bitcoin, вы должны избегать искушения централизованных архитектур, которые могут казаться знакомыми, но в конечном счете, проигрывают безопасности Bitcoin.

Корень доверия

Традиционная архитектура безопасности основывается на концепции называется *корень доверия*, который является доверенным ядром, используемым в качестве основы безопасности всей системы или приложения. Архитектура безопасности развивается вокруг

корня доверия в виде концентрических окружностей, как слои луковицы, расширяя доверие из центра наружу. Каждый слой опирается на более доверенный внутренний слой, используя средства управления доступом, цифровые подписи, шифрование и другие примитивы безопасности. Поскольку программные системы становятся все более сложными, они с большей вероятностью содержат ошибки, которые увеличивают риски нарушения безопасности. В результате, чем более сложна программная система, тем труднее обеспечить ее безопасность. Концепция корня доверия гарантирует, что большая часть доверия находится в наименее сложной части системы, и, следовательно, наименее уязвимой, в то время как более сложное программное обеспечение наслаивается вокруг него. Эта архитектура безопасности повторяется в разных масштабах, установив сначала корень доверия в аппаратной части одной системы, распространяя его посредством операционной системы к системным службам более высокого уровня, и, наконец, через множество слоев и концентрических кругов из серверов.

Архитектура безопасности Bitcoin устроена по-другому. В Bitcoin, система консенсуса создает доверенный публичный реестр, который полностью децентрализован. Правильно подтвержденный blockchain использует блок генезиса в качестве корня доверия, который продолжается в виде цепочки вплоть до текущего блока. Bitcoin-системы могут и должны использовать blockchain в качестве корня доверия. При разработке сложного Bitcoin-приложения, которое состоит из сервисов на многих различных системах, вам следует внимательно изучить архитектуру безопасности для того, чтобы выяснить, где находится доверие. В конечном счете, единственное, что обладает полным доверием — это полностью подтвержденный blockchain. Если ваше приложение явно или неявно наделяет доверием что-то, кроме blockchain, это должно стать источником для беспокойства, поскольку это вносит уязвимость. Для оценки архитектуры безопасности вашего приложения, необходимо рассмотреть каждый отдельный компонент и рассмотреть гипотетический сценарий, при котором этот компонент может быть полностью скомпрометирован и оказаться под контролем взломщика. Возьмите каждый компонент вашего приложения и оцените влияние на общую безопасность, в случае, если этот компонент находится под угрозой. Если ваше приложение перестает быть безопасным, когда некоторые его компоненты скомпрометированы, это показывает, что вы зря передали доверие этим компонентам. Приложение без уязвимостей должно быть уязвимо лишь к компрометации механизма консенсуса Bitcoin, а это означает, что его корень доверия основан на самой сильной части архитектуры безопасности Bitcoin.

Многочисленные примеры взломанных Bitcoin-обменников подчеркивают этот момент. Эти централизованные реализации строят безопасность без blockchain: горячие кошельки, централизованные бухгалтерские базы данных, уязвимые ключи шифрования и другие подобные схемы.

Лучшие практики пользовательской безопасности

Люди использовали физические средства управления безопасностью в течение тысяч лет. Для сравнения, наш опыт работы с цифровой безопасностью составляет менее 50 лет. Современные операционные системы общего назначения, не очень надежны и не особенно подходят для

хранения цифровых денег. Наши компьютеры постоянно подвергаются внешним угрозам через постоянное подключение к Интернету. На них запускаются тысячи программных компонентов от сотен авторов, часто с неограниченным доступом к файлам пользователя. Всего-лишь одна вредоносная компонента, среди многих тысяч установленных на вашем компьютере, может поставить под угрозу клавиатуру и файлы, украсть любые биткоины, хранящиеся в приложениях кошельков. Небольшое количество пользователей обладает необходимыми навыками для содержания компьютера без вирусов и троянов.

Несмотря на десятилетия исследований и достижений в области информационной безопасности, цифровые активы по-прежнему крайне уязвимы против атак. Даже наиболее защищенные и ограниченные системы, в компаниях финансовых услуг, исследований и оборонных подрядчиков, часто бывают взломаны. Bitcoin создает цифровые активы, имеющие внутреннюю ценность и могущие быть украденными и переправленными новым владельцам мгновенно и бесповоротно. Это создает дополнительный стимул для хакеров. До сих пор хакерам приходилось конвертировать идентификационную информацию кредитных карт и банковских счетов в ценность после взломов. Несмотря на сложности с сокрытием и отмыыванием, мы видели возрастающее количество краж. Bitcoin усиливает эту проблему еще больше, потому что его не нужно скрывать или отмыывать; он сам по себе представляет внутреннюю ценность внутри цифрового актива.

К счастью, Bitcoin также создает стимулы для повышения компьютерной безопасности. Если раньше риск взломов казался расплывчатым и косвенным, Bitcoin сделал эти риски понятными и очевидными. Хранение биткоинов на компьютере заставляет пользователей сосредоточиться на необходимости повышения компьютерной безопасности. Как прямой результат распространения и расширения использования Bitcoin и других цифровых валют, мы наблюдаем эскалацию как хакерских методов, так и решений в области безопасности. Проще говоря, у хакеров теперь есть очень сочные цели, а пользователи имеют четкий стимул для самозащиты.

За последние три года, в результате распространения Bitcoin, мы видели огромные инновации в области обеспечения информационной безопасности, аппаратных средств шифрования, хранилищ ключей и аппаратных кошельков, технологий мульти-подписи и цифрового условного депонирования. В следующих разделах мы рассмотрим различные хорошие практики в области пользовательской безопасности.

Физическое хранение Bitcoin

Поскольку большинство пользователей гораздо лучше знакомы с физической безопасностью, чем с информационной, очень эффективным способом защиты биткоинов будет заключаться в преобразовании их в физическую форму. Ключи Bitcoin — это не более, чем длинные числа. Это означает, что они могут быть сохранены в физическом виде, например, напечатаны на бумаге или выгравированы на металлической монете. Безопасность теперь будет заключаться в обеспечении физической защиты печатной копии ключей Bitcoin. Набор ключей Bitcoin, который печатается на бумаге называется "бумажными кошельками", и существует довольно много бесплатных инструментов для их создания. Я лично держу подавляющее большинство

своих биткоинов (99% или более) на бумажных кошельках, зашифрованным с помощью ВІР0038 в виде нескольких копий, запертыми в сейфах. Хранение биткоинов в офлайне называется *холодным хранилищем* и является одним из наиболее эффективных методов с точки зрения безопасности. Для холодного хранения ключи генерируются на системе, находящейся в офлайне (не подключенной к Интернету) и далее они также хранятся в офлайне на бумаге или на цифровых носителях, таких как USB флэш-накопителях.

Аппаратные кошельки

В долгосрочной перспективе, безопасность Биткоин примет форму аппаратных взломоустойчивых кошельков. В отличие от смартфонов или настольных компьютеров, аппаратные кошельки создаются лишь с одной целью: безопасно хранить биткоины. Без ПО общего назначения, через которое такой кошелек можно было бы взломать и благодаря ограниченному интерфейсу, аппаратные кошельки могут предоставить максимальный уровень безопасности для неискушенных пользователей. Я ожидаю, что аппаратные кошельки станут преобладающим способом хранения биткоинов. В качестве примера подобного аппаратного кошелька, взгляните на [Trezor](#).

Балансировка рисков

Несмотря на то, что большинство пользователей справедливо обеспокоены кражами биткоинов, существует еще больший риск. Файлы данных все время теряются. Если они содержат биткоины, потеря гораздо более болезненна. В попытке обезопасить свои Bitcoin-кошельки, пользователи должны быть крайне осторожны, чтобы не зайти слишком далеко, и в конечном итоге не потерять биткоины. В июле 2011 года один известный образовательный Биткоин-проект потерял почти 7000 юиткоинов. Петаясь предотвратить возможные кражи, владельцы создали сложную схему из зашифрованных резервных копий. В конце концов они случайно потеряли ключи шифрования, что сделало резервные копии бесполезными и повлекло потерю целого состояния. Так же, как прятать деньги, закапывая их в пустыне, если вы обезопасите ваши биткоины слишком хорошо, может так случиться, что вы не сможете их получить обратно.

Диверсификация рисков

Вы бы стали носить все свои накопления в кошельке в виде наличных? Большинство людей посчитали бы это безрассудным, но пользователи Bitcoin часто держат все свои биткоины в одном кошельке. Вместо этого пользователи должны распределять риск среди многочисленных и разнообразных кошельков Bitcoin. Предусмотрительные пользователи будут держать лишь малую часть, возможно, менее 5% своих биткоинов в онлайн или мобильном кошельке на "карманные расходы". Остальные средства должны храниться другими способами, лучше всего офлайн, например в холодных хранилищах.

Мультиподпись и управление

Всякий раз, когда какая-нибудь компания или частное лицо желают хранить большое

количество биткоинов, им следует рассмотреть вопрос об использовании мультиподписного адреса. Мультиподписные адреса обезопасивают средства, требуя для их траты более одной подписи. Ключи подписи должны храниться в нескольких различных местах и под контролем различных людей. В корпоративной среде, например, ключи должны быть созданы независимо друг от друга и находиться у нескольких руководителей компании, чтобы всего один человек не мог под угрозу общие средства. Мультиподписные адреса могут также предложить избыточность, когда один человек владеет несколькими ключами, которые могут храниться в разных местах.

Живучесть

Один из важных моментов безопасности, о котором часто забывают, является доступность, особенно в контексте потери трудоспособности или смерти держателя ключа. Пользователям Bitcoin советуют использовать сложные пароли и хранить свои ключи в безопасности и не делиться ими с кем-либо. К сожалению, эта практика делает практически невозможным для семьи пользователя восстановление каких-либо средств, если пользователь не доступен для их разблокирования. В большинстве случаев, на самом деле, семьи пользователей Bitcoin могут быть в полном неведении о существовании средств в Bitcoin.

Если у вас есть много биткоинов, вам следует рассмотреть вопрос об предоставлении деталей доступа доверенному родственнику или адвокату. Более сложная схема живучести может быть составлена с мультиподписным доступом и распоряжением имуществом через адвоката со специализацией в качестве "исполнителя цифровых активов."

Вывод

Bitcoin — это совершенно новая, беспрецедентная и сложная технология. Со временем мы разработаем более эффективные инструменты и методы, которые будут проще в использовании для неспециалистов в безопасности. Пока что пользователи Bitcoin могут использовать многие из советов приведенных здесь, для безопасной и безаварийной работы с Bitcoin.

Appendix A: Команды Bitcoin Explorer (bx)

Использование: bx COMMAND [--help]

Info: Команды bx:

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
```

Для получения дополнительной информации см [Bitcoin Explorer home page](#) и [Bitcoin Explorer user documentation](#).

Примеры использования командны `bx`

Давайте рассмотрим некоторые примеры использования команд Bitcoin Explorer для экспериментов с ключами и адресами:

Создание случайного "зерна" с помощью команды `seed`, которая использует генератор случайных чисел операционной системы. Подадим зерно на вход команды `ec-new` для создания нового приватного ключа. Сохраним стандартный вывод в файл `private_key`:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

Теперь, сгенерируем публичный ключ из этого приватного ключа с помощью команды `ec-to-public`. Подадим файл `private_key` на стандартный ввод и сохраним стандартный вывод команды в новый файл `public_key`:

```
$ bx ec-to-public < private_key > public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

Мы можем переформатировать `public_key` в качестве адреса с помощью команды `ec-to-address`. Мы подадим `public_key` на стандартный ввод:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHYP8Kw7xQad1Lr6XUzWUnkG
```

Ключи, сгенерированные таким образом производят недетерминированный кошелек `type-0`. Это означает, что каждый ключ генерируется из независимого зерна. Команды Bitcoin Explorer также могут генерировать ключи детерминировано, в соответствии с BIP0032. В этом случае "мастер-ключ" создается из зерна, а затем удлиняется детерминировано для создания дерева дочерних ключей, что дает детерминированный кошелек `type-2`.

Сначала мы используем команды `seed` и `hd-new` для создания главного ключа, который будет использоваться в качестве основы для получения иерархии ключей.

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < seed > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgiEjArAVaZaCTgsaGe6LsAnwubeiTcDzd23mAoyizm9cApe51gNfLMkBqkYo
WWMCRwzfuJk8RwF1SVEpAQ
```

Теперь используем команду `hd-private` для создания укрепленного ключа от "счета" и последовательность из двух приватных ключей.

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueufuyQ8Qi3qpjk4aKsbmbfxwcgS8PYbg
oR2NWHeLyvg4DhoEE68A1n

$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnBiDbvG4LgndirDsia1e9F3DW
PkX7Tp1V1u97HKG1FJwUpU

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzbLB4fzHFd6VqCLPGRZFsdsuMVERadbgDbziCRJru9n6tzEWr
ASVpEdrZrFidt1RDfn4yA3
```

Далее мы используем команду `hd-public` для генерирования соответствующей последовательности из двух публичных ключей.

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz
5wzaSW4FiGrseNDR4LKqTy

$ bx hd-public --index 1 < account
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQD
hohEcDFTEYMvYzwrD7Juf8
```

Публичные ключи также могут быть получены из соответствующих приватных ключей, используя команду `hd-to-public`.

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz
5wzaSW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQD
hohEcDFTEYMvYzwrD7Juf8
```

Мы можем сгенерировать практически неограниченное количество ключей в детерминированной цепи, все происходящие от одного зерна. Этот метод используется во многих приложениях-кошельках для генерации ключей, которые могут быть сохранены и восстановлены из резервной копии с помощью единственного значения зерна. Это проще, чем создавать резервную копию кошелька со всеми его случайно сгенерированными ключами каждый раз, когда создается новый ключ.

Зерно может быть закодировано с использованием команды `mnemonic-encode`.

```
$ bx hd-mnemonic < seed > words  
adore repeat vision worst especially veil inch woman cast recall dwell appreciate
```

Зерно затем может быть декодировано с помощью команды mnemonic-decode.

```
$ bx mnemonic-decode < words  
eb68ee9f3df6bd4441a9feadec179ff1
```

Мнемоническое кодирование может сделать зерно легче записываемым и даже запоминаемым.

Appendix A: Предложения по улучшению Bitcoin (Bitcoin Improvement Proposals)

Предложения по улучшению Bitcoin — это проектные документы, содержащие информацию для Bitcoin сообщества, или описывающие новую функцию для Bitcoin или его процессов или окружения.

Согласно BIP0001 *Цели BIP и рекомендации*, есть три вида BIP:

Стандартный BIP

Описывает любое изменение, которое затрагивает большинство или все реализации Bitcoin, такое как изменение сетевого протокола, изменение блока или правил валидации транзакций, или какие-либо другие изменения или дополнения, влияющие на совместимость приложений, использующих Bitcoin.

Информационный BIP

Описывает проблемы проектирования Bitcoin, или содержит общие рекомендации или информацию для Bitcoin-сообщества, но не предлагает новую функцию. Информационные BIP'ы не обязательно представляют собой результат консенсуса Bitcoin сообщества или его прямые рекомендации, так что пользователи и разработчики вольны сами игнорировать информационные BIP'ы либо следовать их советам.

Process BIP

Описывает процесс Bitcoin, или предлагает изменение или событие в процессе. Process BIP подобны стандартным BIP, но применимы и к другим областям, помимо протокола Bitcoin. Они могут предложить реализацию, но не для исходного кода Bitcoin; они часто требуют консенсуса сообщества; и в отличие от информационных BIP, они больше, чем рекомендации, и пользователи, как правило, не могут их игнорировать. Примеры включают в себя процедуры, руководящие принципы, изменения в процессе принятия решений, а также изменения в инструментах или окружении, используемых в развитии Bitcoin. Любой мета-BIP также рассматривается как process BIP.

Предложения по улучшению Bitcoin записываются в версионированном репозитории по адресу [GitHub](#). [Список BIP](#) показывает список BIP'ов на осень 2014 г.

Table 1. Список BIP

BIP#	Ссылка	Название	Автор	Тип	Статус
1	https://github.com/bitcoin/bips/blob/master/bip-0001.mediawiki	Цели BIP и рекомендации	Amir Taaki	Стандарт	Активный

ВІР#	Ссылка	Название	Автор	Тип	Статус
10	https://github.com/bitcoin/bips/blob/master/bip-0010.mediawiki	Распространение Multi-Sig транзакций	Alan Reiner	Информация	Черновик
11	https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki	Стандарт транзакций М-из-N	Gavin Andresen	Стандарт	Принят
12	https://github.com/bitcoin/bips/blob/master/bip-0012.mediawiki	OP_EVAL	Gavin Andresen	Стандарт	Отозван
13	https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki	Формат адреса для pay-to-script-hash	Gavin Andresen	Стандарт	Финал
14	https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki	Версия протокола и User Agent	Amir Taaki, Patrick Strateman	Стандарт	Принят
15	https://github.com/bitcoin/bips/blob/master/bip-0015.mediawiki	Псевдонимы	Amir Taaki	Стандарт	Отозван
16	https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki	Pay To Script Hash	Gavin Andresen	Стандарт	Принят

ВІР#	Ссылка	Название	Автор	Тип	Статус
17	https://github.com/bitcoin/bips/blob/master/bip-0017.mediawiki	OP_CHECKHASHVERIFY (CHV)	Luke Dashjr	Отозван	Черновик
18	https://github.com/bitcoin/bips/blob/master/bip-0018.mediawiki	hashScriptCheck	Luke Dashjr	Стандарт	Черновик
19	https://github.com/bitcoin/bips/blob/master/bip-0019.mediawiki	Стандарт транзакций М-из-N (Low SigOp)	Luke Dashjr	Стандарт	Черновик
20	https://github.com/bitcoin/bips/blob/master/bip-0020.mediawiki	URI Scheme	Luke Dashjr	Стандарт	Заменен
21	https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki	URI Scheme	Nils Schneider, Matt Corallo	Стандарт	Принят
22	https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki	getblocktemplate - Fundamentals	Luke Dashjr	Стандарт	Принят
23	https://github.com/bitcoin/bips/blob/master/bip-0023.mediawiki	getblocktemplate - Майнинг пулы	Luke Dashjr	Стандарт	Принят

ВІР#	Ссылка	Название	Автор	Тип	Статус
30	https://github.com/bitcoin/bips/blob/master/bip-0030.mediawiki	Дубликаты транзакций	Pieter Wuille	Стандарт	Принят
31	https://github.com/bitcoin/bips/blob/master/bip-0031.mediawiki	Pong message	Mike Hearn	Стандарт	Принят
32	https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki	Hierarchical Deterministic Wallets	Pieter Wuille	Информационный	Принят
33	https://github.com/bitcoin/bips/blob/master/bip-0033.mediawiki	Stratized Nodes	Amir Taaki	Стандарт	Черновик
34	https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki	Версия блока v2, Высота в coinbase	Gavin Andresen	Стандарт	Принят
35	https://github.com/bitcoin/bips/blob/master/bip-0035.mediawiki	mempool message	Jeff Garzik	Стандарт	Принят
36	https://github.com/bitcoin/bips/blob/master/bip-0036.mediawiki	Custom Services	Stefan Thomas	Standard	Черновик

ВІР#	Ссылка	Название	Автор	Тип	Статус
37	https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki	Фильтр Блума	Mike Hearn and Matt Corallo	Стандарт	Принят
38	https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki	Защищенные парольной фразой приватные ключи	Mike Caldwell	Стандарт	Черновик
39	https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki	Мнемонический код для генерирования детерминированных ключей	Slush	Стандарт	Черновик
40		Stratum wire protocol	Slush	Стандарт	Присвоен номер ВІР
41		Stratum mining protocol	Slush	Стандарт	Присвоен номер ВІР
42	https://github.com/bitcoin/bips/blob/master/bip-0042.mediawiki	Конечная денежная эмиссия для Биткоин	Pieter Wuille	Стандарт	Черновик
43	https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki	Purpose Field for Deterministic Wallets	Slush	Стандарт	Черновик
44	https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki	Multi-Account Hierarchy for Deterministic Wallets	Slush	Стандарт	Черновик

ВІР#	Ссылка	Название	Автор	Тип	Статус
50	https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki	March 2013 Chain Fork Post-Mortem	Gavin Andresen	Информационный	Черновик
60	https://github.com/bitcoin/bips/blob/master/bip-0060.mediawiki	Fixed Length "version" Message (Relay-Transactions Field)	Amir Taaki	Стандарт	Черновик
61	https://github.com/bitcoin/bips/blob/master/bip-0061.mediawiki	"reject" P2P message	Gavin Andresen	Стандарт	Черновик
62	https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki	Dealing with malleability	Pieter Wuille	Стандарт	Черновик
63		Stealth-адреса	Peter Todd	Стандарт	Присвоен номер ВІР
64	https://github.com/bitcoin/bips/blob/master/bip-0064.mediawiki	getutxos message	Mike Hearn	Стандарт	Черновик
70	https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki	Payment protocol	Gavin Andresen	Стандарт	Черновик

ВІР#	Ссылка	Название	Автор	Тип	Статус
71	https://github.com/bitcoin/bips/blob/master/bip-0071.mediawiki	Payment protocol MIME types	Gavin Andresen	Стандарт	Черновик
72	https://github.com/bitcoin/bips/blob/master/bip-0072.mediawiki	Payment protocol URIs	Gavin Andresen	Стандарт	Черновик
73	https://github.com/bitcoin/bips/blob/master/bip-0073.mediawiki	Use "Accept" header with Payment Request URLs	Stephen Pair	Standard	Draft

Appendix A: pycoin, ku, и tx

Библиотека [pycoin](#) для языка программирования Питон, написана и поддерживается Ричардом Киссом. Эта библиотека поддерживает манипуляции с ключами Bitcoin и транзакциями, и даже поддерживает язык сценариев в достаточной степени, чтобы правильно разобрать нестандартные транзакции.

Библиотека `pycoin` поддерживает как Python 2 (2.7.x), так и Python 3 (после 3.3), и поставляется с некоторыми удобными утилитами командной строки: `ku` и `tx`.

Key Utility (KU)

Утилита командной строки `ku` ("key utility") — это швейцарский нож для манипулирования ключами. Он поддерживает BIP32-ключи, WIF и адреса (биткойна и альткойнов). Ниже приведены некоторые примеры.

Создать ключ BIP32 используя источники энтропии по умолчанию GPG и `/dev/random`:

```
$ ku create
```

```
input           : create
network         : Bitcoin
wallet key      : xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5Su9DcWHvXJemiJBsY7VqXUG7hipgdWaU
                  m2nhnzdvxJf5KJo9vjP2nABX65c5sFsVsV8oXcbpehtJi
public version  : xpub661MyMwAqRbcFpYYiuvZpKjKhJJDZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                  DGcPfT56AMFeo8M8KPkFMfLUtvjwb6WPv8rY65L2q8Hz
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   :
76460638240546478364843397478278468101877117767873462127021560368290114016034
public pair y   :
59807879657469774102040120298272207730921291736633247737077406753676825777701
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 : 1FNRRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
uncompressed    : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

Создать ключ VIP32 из ключевой фразы:

WARNING

Фраза в этом примере слишком легко угадывается.

```
$ ku P:foo
```

```
input          : P:foo
network        : Bitcoin
wallet key     : xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
                ZoY5eSJMj2Vbyvi2hbmQnCuHbujZ2WXGTux1X2k9Krdtq
public version : xpub661MyMwAqRbcFVF9ULcQLdsEa5WnCCugQAacgNd9iEMQ31tgH6u4DLQWoQayvtS
                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":
"04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "public_pair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
  "wallet_key":
"xpub661MyMwAqRbcFVF9ULcQLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy",
  "chain_code": "5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
  "child_index": "0",
  "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
  "btc_address": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
  "fingerprint": "5d353a2e",
  "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
  "input": "P:foo",
  "public_pair_x":
"81821982719381104061777349269130419024493616650993589394553404347774393168191",
  "public_pair_y":
"58994218069605424278320703250689780154785099509277691723126325051200459038290",
  "key_pair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```

Публичный ключ VIP32:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcQLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
```

Сгенерировать дочерний ключ:

```
$ ku -w -s3/2 P:foo
xprv9wTErTSkjVyJa1v4cUTFMFkWMMe5eu8ErbQcs9xajnsUzCBT7ykHAWdrxvG3g3f6BFk7ms5hHBvmbdutNmyg6i
ogWKxx6mefEw4M8EroLgKj
```

Укрепленный дочерний ключ:

```
$ ku -w -s3/2H P:foo
xprv9wTErTSu5AWGkDeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3DhsuaGZbDh8iDk
BN45k67UKsJUXM1JfRCdn1
```

WIF:

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

Адрес:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAi i
```

Создать несколько дочерних ключей:

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYnSekc8bYuJS8G3bhX
xKWB89Ggn2dzLcoJsuEdRK
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7WJZPnYAmvc77gUQVeJ5QHckc5Adtwxa28ACmANi9XhCrRvtFqQcUxt8r
UgFz3souMiDdWxJDZnQxxz
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxa6srXUPBtj3PTxQFkZJAiwoUpmvtrxKZu
4zfsnr3pqqy2vthpkwuovq
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2jtFXdiEY8UsRNJfqK6DAd
yZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDfyBXmZw4jEYXUHYc9fT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XFKkSsaYKWKJbR5
4bnyAD9GzjUYbAYTtN4ruo
```

Создать соответствующие адреса:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkj dHnPUdLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmc inzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVEM
```

Создать соответствующие WIF:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3yqQxK6zH2NQGxLDee2H9v4Lvwg14cLJW7QwWPzCtKHdWMAQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnMTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFSgwCyuuVFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Убедитесь, что это работает, выбрав VIP32-строку (такую, которая бы соответствовала дочернему ключу 0/3):

```
$ ku -W
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnMTDMrqemY8UF
$ ku -a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmc inzu4aaPdrYqvuiBEK
```

Да, выглядит знакомо.

Из секретной экспоненты:

\$ 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
```

Dogecoin WIF:

```
$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qr ioRbQmjxac5TVoTtZuot
```

Из пары публичных (в Testnet):

```

$ ku -nT
55066263022277343669578718895168534326250603453777594175500187360389116729240,even

input          :
550662630222773436695787188951685343262506034537775941755001873603
                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 Dogecoin-адреса:

```
$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6
```

```
input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Dogecoin
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE
```

Утилита для работы с транзакциями (TX)

Утилита командной строки tx отображает транзакции в читабельном виде, запрашивает базовые транзакции из кэша транзакций pycoin или из веб-сервисов (в настоящее время поддерживаются blockchain.info, blockr.io, и biteasy.com), объединяет транзакции, добавляет или удаляет входы или выходы, и подписывает сделки.

Ниже приведены некоторые примеры.

Посмотреть знаменитую транзакцию "покупка пиццы" [PIZZA]:

```
$ 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:BLOCKEXPLORER
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
```

Упс, мы не настроили веб-сервисы. Давайте сделаем это сейчас:

```
$ PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
$ export PYCOIN_CACHE_DIR PYCOIN_SERVICE_PROVIDERS
```

Это не делается автоматически, так что утилита командной строки не сможет передать на вебсайт третьих лиц потенциально конфиденциальную информацию о том, какие транзакции

вас интересуют. Если вам это безразлично, можете поместить эти строки в свой файл *.profile*.

Попробуем снова:

```
$ 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
010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a4
93046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e
9199238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901fffffffff010010a5d4e8000
0001976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

** can't validate transaction as source transactions missing
```

Последняя строка указывает, что для проверки подписей транзакций технически нужен исходный код транзакции. Поэтому давайте предоставим эту информацию с помощью ключа -a:

```

$ 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 propagate
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

010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a4
93046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e
9199238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901fffffffff010010a5d4e8000
0001976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

all incoming transaction values validated

```

Теперь давайте посмотрим на неизрасходованные выходы для конкретного адреса (UTXO). В блоке #1, мы видим coinbase-транзакцию на 12c6DSiU4Rq3P4ZxziKxZrL5LmMBrzjrJX. Давайте используем `fetch_unspent`, чтобы найти все монеты по этому адресу:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/31/76a914119b098e2e9
80a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/86/76a914119b098e2e9
80a229e139a9ed01a469e518e6f2688ac/10000
a66dddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cfe68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfda4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/5/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/1/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/100000
fd87f9adebb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/1
dfd0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/2000000

0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b538e853519c
726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e0a604f8141781e622947
21166bf621e73a82cbf2342c858eeac/5000000000
```

Appendix A: Операторы, константы и символы языка транзакций

В [Добавить значение в стек](#) показаны операторы для добавления элемента в стек.

Table 1. Добавить значение в стек

Символ	Значение (шестнадцатеричное)	Описание
OP_0 or OP_FALSE	0x00	В стек помещается пустой массив
1-75	0x01-0x4b	Поместить следующие N байтов в стек, где N — это от 1 до 75 байтов
OP_PUSHDATA1	0x4c	Следующий байт скрипта содержит N, поместить следующие N байтов в стек
OP_PUSHDATA2	0x4d	Следующие 2 байта скрипта содержат N, поместить следующие N байтов в стек
OP_PUSHDATA4	0x4e	Следующие 4 байта скрипта содержат N, поместить следующие N байтов в стек
OP_1NEGATE	0x4f	Поместить в стек значение "-1"
OP_RESERVED	0x50	Останов - ошибочная транзакция в случае, если команда не находится в теле невыполненного блока OP_IF
OP_1 or OP_TRUE	0x51	Поместить в стек значение "1"
OP_2 to OP_16	0x52 to 0x60	Для любого из OP_N, поместить в стек значение "N" onto the stack. Например, OP_2 поместит значение "2"

В [Условное управление потоком выполнения](#) показаны операторы управления потоком выполнения.

Table 2. Условное управление потоком выполнения

Символ	Значение (шестнадцатеричное)	Описание
OP_NOP	0x61	Ничего не делать
OP_VER	0x62	Останов - ошибочная транзакция в случае, если команда не находится в теле невыполненного блока OP_IF
OP_IF	0x63	Выполнить последующие команды, если на вершине стека находится не 0
OP_NOTIF	0x64	Выполнить последующие команды, если на вершине стека находится 0
OP_VERIF	0x65	Останов - Ошибочная транзакция
OP_VERNOTIF	0x66	Останов - Ошибочная транзакция
OP_ELSE	0x67	Выполнить только если предыдущие выражения не были выполнены
OP_ENDIF	0x68	Конец блоков OP_IF, OP_NOTIF, OP_ELSE
OP_VERIFY	0x69	Проверить вершину стека, остановить и объявить транзакцию ошибочной в случае, если не TRUE
OP_RETURN	0x6a	Останов и объявление транзакции ошибочной

В [Операции над стеком](#) показаны операторы манипулирования стеком.

Table 3. Операции над стеком

Символ	Значение (шестнадцатеричное)	Описание
OP_TOALTSTACK	0x6b	Извлечь верхний элемент из стека и поместить в альтернативный стек
OP_FROMALTSTACK	0x6c	Извлечь верхний элемент из альтернативного стека и поместить в основной стек

Символ	Значение (шестнадцатеричное)	Описание
OP_2DROP	0x6d	Вынуть два элемента из стека
OP_2DUP	0x6e	Продублировать два верхних элемента стека
OP_3DUP	0x6F	Продублировать три верхних элемента стека
OP_2OVER	0x70	Скопировать третий и четвертый элементы на вершину стека
OP_2ROT	0x71	Переместить пятый и шестой элементы на вершину стека
OP_2SWAP	0x72	Поменять местами две верхние пары элементов в стеке
OP_IFDUP	0x73	Продублировать верхний элемент стека, если это не 0
OP_DEPTH	0x74	Посчитать элементы в стеке и поместить результат в стек
OP_DROP	0x75	Извлечь верхний элемент стека
OP_DUP	0x76	Продублировать верхний элемент в стеке
OP_NIP	0x77	Извлечь второй элемент стека
OP_OVER	0x78	Скопировать второй элемент стека и поместить его на вершину
OP_PICK	0x79	Скопировать N-ый элемент на вершину стека
OP_ROLL	0x7A	Переместить N-ый элемент на вершину стека
OP_ROT	0x7b	Чередовать (налево) три верхних элемента
OP_SWAP	0x7c	Поменять местами два элемента в стеке
OP_TUCK	0x7d	Скопировать верхний элемент и вставить его между верхним и вторым элементом.

В [Операции со строками](#) показаны строковые операторы.

Table 4. Операции со строками

Символ	Значение (шестнадцатеричное)	Описание
OP_CAT	0x7e	Отключен (Конкатинирует два элемента на вершине стека)
OP_SUBSTR	0x7F	Отключен (вернуть подстроку)
OP_LEFT	0x80	Отключен (возвращает подстроку слева)
OP_RIGHT	0x81	Отключен (возвращает подстроку справа)
OP_SIZE	0x82	Подсчитать длину строки на вершине стека и поместить результат в стек

В [Бинарная арифметика и условия](#) показаны операторы двоичной арифметики и булевой логики.

Table 5. Бинарная арифметика и условия

Символ	Значение (шестнадцатеричное)	Описание
OP_INVERT	0x83	Отключен (Инвертировать биты верхнего элемента)
OP_AND	0x84	Отключен (логическое И двух верхних элементов)
OP_OR	0x85	Отключен (логическое ИЛИ двух верхних элементов)
OP_XOR	0x86	Отключен (логической ВЗАИМОИСКЛЮЧАЮЩЕЕ ИЛИ двух верхних элементов)
OP_EQUAL	0x87	Поместить TRUE (1), если два верхних элемента точно равны, иначе поместить FALSE (0)

Символ	Значение (шестнадцатеричное)	Описание
OP_EQUALVERIFY	0x88	То же, что OP_EQUAL, но с последующим запуском OP_VERIFY для останова в случае если не TRUE
OP_RESERVED1	0x89	Останов - ошибочная транзакция в случае, если команда не находится в теле невыполненного блока OP_IF
OP_RESERVED2	0x8a	Останов - ошибочная транзакция в случае, если команда не находится в теле невыполненного блока OP_IF

В [Операции с числами](#) показаны арифметические операторы.

Table 6. Операции с числами

Символ	Значение (шестнадцатеричное)	Описание
OP_1ADD	0x8b	Прибавить 1 к верхнему элементу
OP_1SUB	0x8c	Вычесть 1 от верхнего элемента
OP_2MUL	0x8d	Отключен (умножить верхний элемент на 2)
OP_2DIV	0x8e	Отключен (разделить верхний элемент на 2)
OP_NEGATE	0x8f	Поменять знак верхнего элемента на противоположный
OP_ABS	0x90	Поменять знак верхнего элемента на плюс
OP_NOT	0x91	Если верхний элемент равен 0 или 1 логически инвертировать, иначе вернуть 0
OP_0NOTEQUAL	0x92	Если верхний элемент равен 0 вернуть 0, в противном случае вернуть 1

Символ	Значение (шестнадцатеричное)	Описание
OP_ADD	0x93	Извлечь два верхних элемента, сложить их и поместить результат в стек
OP_SUB	0x94	Извлечь два верхних элемента, вычесть первое из второго, поместить результат в стек
OP_MUL	0x95	Отключен (перемножить два верхних элемента)
OP_DIV	0x96	Отключен (поделить второй элемент на значение из второго элемента)
OP_MOD	0x97	Отключен (остаток от деления второго элемента на содержимое первого элемента)
OP_LSHIFT	0x98	Отключен (побитовый сдвиг влево второго элемента на количество битов из первого элемента)
OP_RSHIFT	0x99	Отключен (побитовый сдвиг вправо второго элемента на количество битов из первого элемента)
OP_BOOLAND	0x9A	Логическое И двух верхних элементов
OP_BOOLOR	0x9b	Логическое ИЛИ двух верхних элементов
OP_NUMEQUAL	0x9C	Возвращает TRUE, если два верхних элемента представляют собой одинаковые числа
OP_NUMEQUALVERIFY	0x9d	То же, что NUMEQUAL, но с OP_VERIFY и остановом, если не TRUE
OP_NUMNOTEQUAL	0x9e	Вернуть TRUE, если два верхних элемента не равные числа

Символ	Значение (шестнадцатеричное)	Описание
OP_LESSTHAN	0x9F	Вернуть TRUE, если второй элемент меньше верхнего элемента
OP_GREATERTHAN	0xa0	Вернуть TRUE, если второй элемент больше, чем верхний элемент
OP_LESSTHANOREQUAL	0xa1	Вернуть TRUE, если второй элемент меньше или равен верхнему элементу
OP_GREATERTHANOREQUAL	0xa2	Вернуть TRUE, если второй пункт больше или равен верхнему элементу
OP_MIN	0xa3	Вернуть меньшее из двух верхних элементов
op_max	0xa4	Вернуть больший из двух верхних элементов
OP_WITHIN	0xA5	Вернуть TRUE, если значение третьего элемента между (или равно) вторым и первым элементами

В [Криптографические операторы и операции хэширования](#) показаны криптографические операторы.

Table 7. Криптографические операторы и операции хэширования

Символ	Значение (шестнадцатеричное)	Описание
OP_RIPEMD160	0xa6	Вернуть хэш RIPEMD160 верхнего элемента
OP_SHA1	0xa7	Вернуть SHA1-хэш верхнего элемента
OP_SHA256	0xa8	Вернуть SHA256-хэш верхнего элемента
OP_HASH160	0xa9	Вернуть RIPEMD160(SHA256(x)) хэш верхнего элемента
OP_HASH256	0xAA	Вернуть SHA256(SHA256(x)) хэш верхнего элемента

Символ	Значение (шестнадцатеричное)	Описание
OP_CODESEPARATOR	0xab	Отмечает начало данных для проверки подписей
OP_CHECKSIG	0xac	Извлечь публичный ключ и подпись и проверить подпись на хешированных данных транзакции, вернуть TRUE, если проверка удалась
OP_CHECKSIGVERIFY	0xad	То же, что CHECKSIG, но с последующим OP_VERIFY и остановом, если не TRUE
OP_CHECKMULTISIG	0xae	Запустить CHECKSIG для каждой предоставленной пары подписи и публичного ключа. Все должны совпасть. Ошибка в реализации помещает в стек дополнительное значение, обходится при помощи OP_NOP
OP_CHECKMULTISIGVERIFY	0xAF	То же, что CHECKMULTISIG, но с последующим OP_VERIFY и остановом в случае, если не TRUE

В [Неоператоры](#) показаны неоператорные символы

Table 8. Неоператоры

Символ	Значение (шестнадцатеричное)	Описание
OP_NOP1-OP_NOP10	0xb0-0xb9	Ничего не делает, игнорируется

В [OP-коды, зарезервированные для внутреннего использования парсером](#) показаны коды операторов, зарезервированных для использования внутри парсера.

Table 9. OP-коды, зарезервированные для внутреннего использования парсером

Символ	Значение (шестнадцатеричное)	Описание
OP_SMALLDATA	0xf9	Представляет небольшое поле данных

Символ	Значение (шестнадцатеричное)	Описание
OP_SMALLINTEGER	0xfa	Представляет небольшое целое поле данных
OP_PUBKEYS	0xfb	Представляет поля публичных ключей
OP_PUBKEYHASH	0xfd	Представляет поле хеша публичного ключа
OP_PUBKEY	0xfe	Представляет поле публичного ключа
OP_INVALIDOPCODE	0xff	Представляет любой неназначенный OP-код