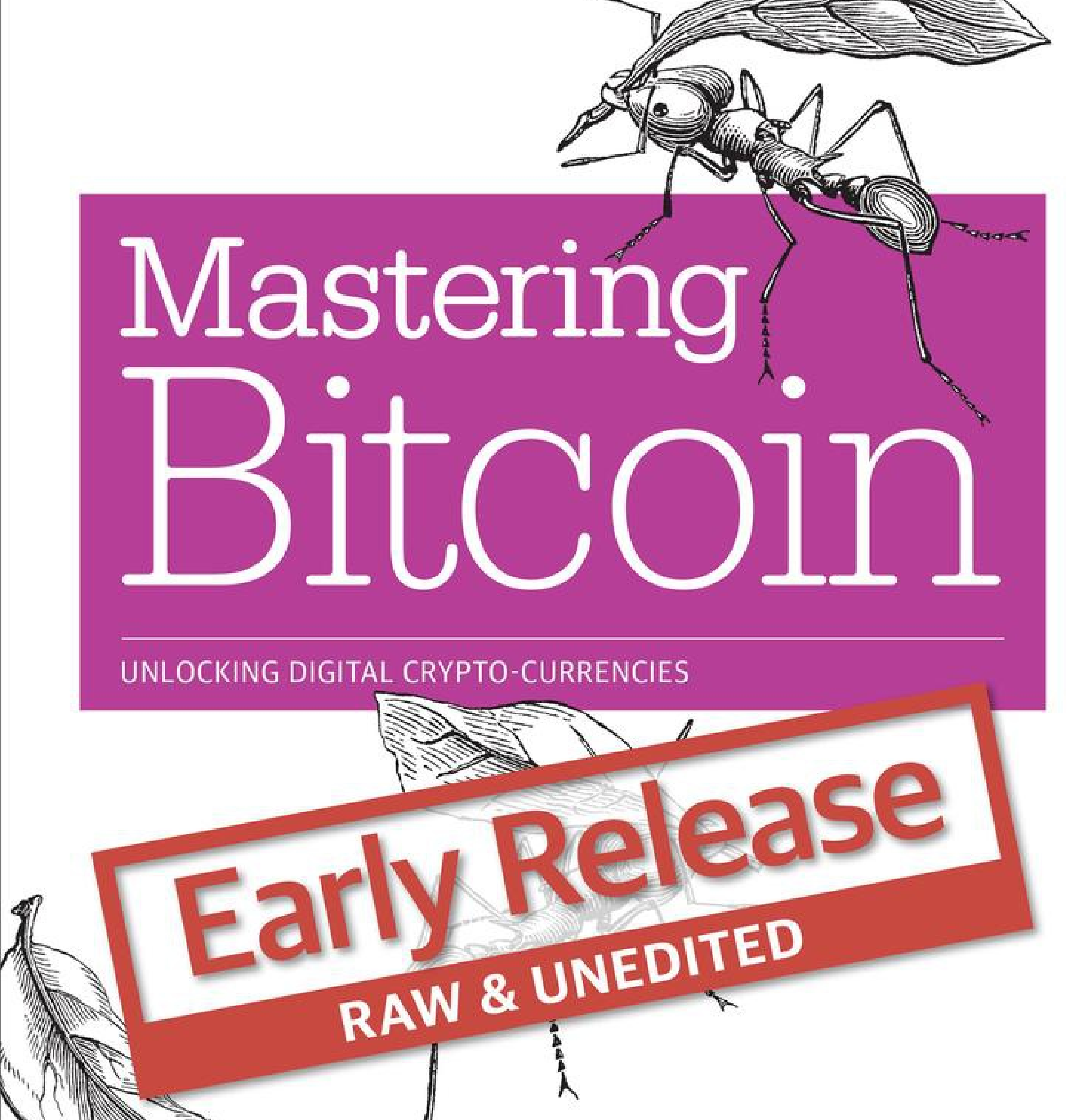
O'REILLY®

Mastering Bitcoin

UNLOCKING DIGITAL CRYPTO-CURRENCIES

Early Release

RAW & UNEDITED



Andreas M. Antonopoulos

Mastering Bitcoin

Andreas M. Antonopoulos

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Preface

Writing the Bitcoin Book

I first stumbled upon bitcoin in mid-2011. My immediate reaction was more or less “Pfft! Nerd money!” and I ignored it for another 6 months, failing to grasp its importance. This is a reaction which I have seen repeated among many of the smartest people I know, which gives me some consolation. The second time I came across bitcoin in a mailing list discussion, I decided to read the white paper written by Satoshi Nakamoto, to study the authoritative source and see what it was all about. I still remember the moment I finished reading those 9 pages, when I realized that bitcoin was not simply a digital currency, but a network of trust that could also provide the basis for so much more than just currencies. That realization: “This isn’t money, it’s a de-centralized trust network,” started me on a four month journey to devour every scrap of information about bitcoin I could find. I became obsessed and enthralled, spending twelve or more hours each day glued to a screen, reading, writing, coding and learning as much as I could. I emerged from this state of fugue, more than 20 lbs lighter from lack of consistent meals, determined to dedicate myself to working on bitcoin.

Two years later, after creating a number of small startups to explore various bitcoin-related services and products, I decided that it was time to write my first book. Bitcoin was the topic that had driven me into a frenzy of creativity, consumed my thoughts and is the most exciting technology I have encountered since the Internet. It was now time to share my discovery of this amazing technolgy and my passion with a broader audience. This is the bitcoin book.

Intended Audience

This book is mostly intended for coders. If you can use a programming language, this book will teach you how cryptographic currencies work, how to use them and how to develop software that works with them. The first few chapters are also suitable as an in-depth introduction to bitcoin for non-coders - those trying to understand the inner workings of bitcoin and cryptocurrencies. The examples are illustrated in Python and on the command-line of a Unix-like operating system such as Linux.

Early-Release Note

The early release version of the book is a **raw and rough draft** and will change regularly. New chapters will be added as they are drafted and there will be plenty of changes to the content, examples and diagrams. There will be factual and technical errors in the early release and some of the examples may not work or refer to obsolete versions of the code. Nevertheless, I hope you will enjoy the content and find it useful. I also hope that you will take the opportunity to “fork” the source code of the book and provide feedback by creating a pull request or submitting a patch. I present this work in the spirit of Cunningham’s Law, named after the inventor of the wiki, Ward Cunningham:

The best way to get the right answer on the Internet is not to ask a question, it's to post the wrong answer

I hope you can help me find and publish the "right answer" by the time this book is ready to print.

Why Are There Bugs On The Cover?

The Leafcutter Ant is a species that exhibits highly complex behavior in a colony super-organism, but each individual ant operates on a set of simple rules driven by social interaction and the exchange of chemical scents (pheromones). Per Wikipedia: "Next to humans, leafcutter ants form the largest and most complex animal societies on Earth." Leafcutter ants don't actually eat leaves, but rather use them to farm a fungus, which is the central food source for the colony. Get that? These ants are farming!

While ants form a caste-based society and have a queen for producing offspring, there is no central authority or leader in an ant colony. The highly intelligent and sohpisticated behavior exhibited by a multi-million member colony is an emergent property from the interaction of the individuals in a social network.

Nature demonstrates that de-centralized systems can be resilient and can produce emergent complexity and sophistication without the need for a central authority, hierarchy or complex parts.

Bitcoin is a highly sophisticated de-centralized trust network that can support a myriad of financial processes. Yet, each node in the bitcoin network follows a few simple mathematical rules. The interaction between many nodes is what leads to the emergence of the sophisticated behavior, not any inherent complexity or trust in any single node. Like an ant colony, the bitcoin network is a resilient network of simple nodes following simple rules that together can do amazing things without any central coordination.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width bold
: Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This icon signifies a tip, suggestion, or general note.

WARNING

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Book Title* by Some Author (O'Reilly). Copyright 2012 Some Copyright Holder, 978-0-596-xxxx-x."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones

& Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:
O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://shop.oreilly.com/product/0636920032281.do.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly

Follow us on Twitter: http://twitter.com/oreillymedia

Watch us on YouTube: http://www.youtube.com/oreillymedia

Quick Glossary

This quick glossary contains many of the terms used in relation to bitcoin. These terms are used throughout the book, so bookmark this for a quick reference and clarification.

address (aka public key)
A bitcoin address looks like 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV, they always start with a one. You can have as many as you like, share them so people can send you coins.

bitcoin
The name of the currency unit (the coin), the network and the software

block
A grouping of transactions, marked with a timestamp, and a fingerprint of the previous block. The block header is hashed to find a proof-of-work, thereby validating the transactions. Valid blocks are added to the main blockchain by network consensus.

blockchain
A list of validated blocks, each linking to its predecessor all the way to the genesis block.

confirmations

Once a transaction is included in a block, it has “one confirmation”. As soon as *another* block is mined on the same blockchain, the transaction has two confirmations etc. Six or more confirmations is considered final.

difficulty

A network-wide setting that controls how much computation is required to find a proof-of-work.

difficulty target

A difficulty at which all the computation in the network will find blocks approximately every 10 minutes.

difficulty re-targetting

A network-wide re-calculation of the difficulty which occurs once every 2106 blocks and considers the hashing power of the previous 2106 blocks.

fees

An excess amount included in each transaction as a network fee or additional reward to the miner who finds the proof-of-work for the new block. Currently 0.5 mBTC minimum.

hash

A digital fingerprint of some binary input.

genesis block

The first block in the blockchain, used to initialize the crypto-currency

miner

A network node that finds valid proof-of-work for new blocks, by repeated hashing

network

A peer-to-peer network that propagates transactions and blocks to every bitcoin node on the network.

proof-of-work

A piece of data that requires significant computation to find. In bitcoin, miners must find a numeric solution to the SHA256 algorithm that meets a network wide target, the difficulty target.

reward

An amount included in each new block as a reward by the network to the miner who found the proof-of-work solution. It is currently 25BTC per block.

secret key (aka private key)

The secret number that unlocks bitcoins sent to the corresponding address.

transaction

In simple terms, a transfer of bitcoins from one address to another. More precisely, a transaction is a signed data structure expressing a transfer of value. Transactions are transmitted over the bitcoin network, collected by miners and included into blocks, made permanent on the blockchain.

wallet

Software that holds all your addresses. Use it to send bitcoin and manage your keys.

Chapter 1. Introduction

What is Bitcoin?

Bitcoin is collection of concepts and technologies that form the basis of a digital money ecosystem. It includes a currency, with units called bitcoins, that are used to store and transmit value among participants in the bitcoin network. Bitcoin users communicate with each other using the bitcoin protocol, primarily via the Internet, although other transport networks can also be used. The bitcoin protocol stack, available as open source software, can be run on a wide range of computing devices, including laptops and smartphones, making the technology easily accessible.

Users can transfer bitcoin over the network to do just about anything that can be done with conventional currencies, such as buy and sell goods, send money to people or organizations, or extend credit. Bitcoin technology includes features, based on encryption and digital signatures, to ensure the security of the bitcoin network. Bitcoins can be purchased and sold, exchanged for other currencies at a floating exchange rate, at specialized currency exchanges. Bitcoin in a sense is the perfect form of money for the Internet: fast, secure, borderless.

Unlike traditional currencies, bitcoins are entirely virtual. There are no physical coins, or even digital coins per se. The coins are implied in transactions which transfer value from sender to recipient. Users of bitcoin own keys which allow them to prove ownership of transactions in the bitcoin network, unlocking the value to spend it and transfer it to a new recipient. Those keys are stored in a digital wallet on each user’s computer. Possession of the key that unlocks a transaction is the only prerequisite to spending bitcoins, putting the control entirely in the hands of each user.

Bitcoin is a fully-distributed, peer-to-peer system, and as such there is no “central” server or point of control. Bitcoins are created through a process called “mining”, which involves looking for a solution to a difficult problem. Any participant in the bitcoin network (i.e. any device running the full bitcoin protocol stack) may operate as a miner, using their computer’s processing power to attempt to find solutions to this problem. Every 10 minutes on average, a new solution is found by someone who then is able to validate the transactions of the past 10 minutes and is rewarded with brand new bitcoins. Essentially, the currency-issuance function of a central bank and the clearing function are de-centralized and turned into a global competition.

The bitcoin protocol includes built-in algorithms that regulate the mining function across the network. The difficulty of the problem that miners must solve is adjusted dynamically so that, on average, someone finds a correct answer every 10 minutes regardless of how many miners (and CPUs) are working on the problem at any moment. The protocol also halves the rate at which new bitcoins are created every 4 years, and limits the total number of bitcoins that will be created to a fixed total of 21 million coins. The result is that the number of bitcoins in circulation closely follows an easily predictable curve that reaches 21 million by the year 2140. As a result, the bitcoin currency is deflationary and cannot be inflated by “printing” new money above and beyond the expected issuance rate.

Behind the scenes, bitcoin is also the name of protocol, a network and a distributed computing innovation. The bitcoin currency is really only the first application of this invention. As a developer, I see bitcoin as akin to the Internet of money, a network for propagating value and securing the ownership of digital assets via distributed computation. There's a lot more to bitcoin than first meets the eye.

In this chapter we'll get started by explaining some of the main concepts and terms, getting the necessary software and using bitcoin for simple transactions. In following chapters we'll start unwrapping the layers of technology that make bitcoin possible and examine the inner workings of the bitcoin network and protocol.

History of Bitcoin

The emergence of viable digital money is closely linked to developments in cryptography. This is not surprising when one considers the fundamental challenges involved with using bits to represent value that can be exchanged for goods and services. Two fundamental questions for anyone accepting digital money, are:

1. Can I trust the money is authentic and not counterfeit?
2. Can I be sure that no one else can claim that this money belongs to them and not me? (aka the "double-spend" problem)

Issuers of paper money are constantly battling the counterfeiting problem, by using increasingly sophisticated papers and printing technology. Physical money addresses the double-spend issue easily because the same paper note cannot be in two places at once. Of course, conventional money is also often stored and transmitted digitally. In this case the counterfeiting and double-spend issues are handled by clearing all electronic transactions through central authorities that have a global view of the currency in circulation. For digital money, which cannot take advantage of esoteric inks or holographic strips, cryptography provides the basis for trusting the legitimacy of a user's claim to value. Specifically, cryptographic digital signatures enable a user to sign a digital asset or transaction proving the ownership of that asset. With the appropriate architecture, digital signatures also can be used to address the double-spend issue.

In the late 1980s, when cryptography started becoming more broadly available and understood, many researchers began trying to use cryptography to build digital currencies. These early digital currency projects issued digital money, usually backed by a national currency or precious metal such as gold.

While these earlier digital currencies worked, they were centralized and as a result they were easy to attack by governments and hackers. Early digital currencies used a central clearinghouse to settle all transactions at regular intervals, just like a traditional banking system. Unfortunately, in most cases these nascent digital currencies were targeted by worried governments and eventually litigated out of existence. Some failed in spectacular crashes when the parent company liquidated abruptly. To be robust against intervention by antagonists, be they legitimate governments or criminal elements, a digital currency needed to avoid the use of a central currency issuing or transaction clearing authority that could be a single point of attack. Bitcoin is such a system, completely de-centralized by design, lacking any central authority or point of control that can be attacked or corrupted.

Bitcoin represents the culmination of decades of research in cryptography and distributed systems and includes four key innovations brought together in a unique and powerful combination. Bitcoin consists of:

- A de-centralized peer-to-peer network (the bitcoin protocol);
- A public transaction ledger (the blockchain);
- A de-centralized mathematical and deterministic currency issuance (distributed mining), and;
- A de-centralized transaction verification system (transaction script)

Bitcoin was invented in 2008 by Satoshi Nakamoto with the publication of a paper titled "Bitcoin: A Peer-to-Peer Electronic Cash System". Satoshi Nakamoto combined several prior inventions such as b-money and HashCash to create a completely de-centralized electronic cash system that does not rely on a central authority for settlement and validation of transactions. The key innovation was to use a Proof-Of-Work algorithm to conduct a global "election" every 10 minutes, allowing the de-centralized network to arrive at *consensus* about the state of transactions. This elegantly solves the issue of double-spend, a weakness of digital money, where a single currency unit can be spent twice. Previously, the double-spend problem was solved by clearing all transactions through a central clearinghouse.

The bitcoin network started in 2009, based on a reference implementation published by Nakamoto and since revised by many other programmers. During the first four years of operation, the network has grown to include an enormous amount of Proof-Of-Work computation, thereby increasing its security and resilience. In 2013, the total market value of bitcoin's primary monetary supply measure (M0) is estimated at more than 10 billion US dollars. The largest transaction processed by the network was $150 million US dollars, transmitted instantly and processed without any fees.

Satoshi Nakamoto withdrew from the public in April of 2011, leaving the responsibility of developing the code and network to a thriving group of volunteers. The name Satoshi Nakamoto is an alias and the identity of the person or people behind this invention is currently unknown. However, neither Satoshi Nakamoto nor anyone else exerts control over the bitcoin system, which operates based on fully transparent mathematical principles. The invention itself is groundbreaking and has already spawned new science in the fields of distributed computing, economics and econometrics.

A SOLUTION TO A DISTRIBUTED COMPUTING PROBLEM

Satoshi Nakamoto's invention is also a practical solution to a previously unsolved problem in distributed computing, known as the Byzantine Generals problem. Briefly, the problem consists of trying to agree on a course of action by exchanging information over an unreliable and potentially compromised network. Satoshi Nakamoto's solution, which uses the concept of Proof-of-Work to achieve consensus without a central trusted authority represents a breakthrough in distributed computing science and has wide applicability beyond currency. It can be used to achieve consensus on decentralized networks for provably-fair elections, lotteries, asset registries, digital notarization and more.

Bitcoin Uses, Users and Their Stories

Bitcoin is a technology, but it expresses money which is fundamentally a language for exchanging value between people. Let’s look at the people who are using bitcoin and some of the most common uses of the currency and protocol through their stories. We will re-use these stories throughout the book to illustrate the real-life uses of digital money and how they are made possible by the various technologies that are part of bitcoin.

North American Retail

Alice lives in Northern California, in the Bay Area. She has heard about bitcoin from her techie friends and wants to start using it. We will follow her story as she learns about bitcoin, acquires some and then spends some of her bitcoin to buy a cup of coffee at Bob’s Cafe in Palo Alto. This story will introduce us to the software, the exchanges and basic transactions from the perspective of a retail consumer.

Offshore Contract Services

Bob, the cafe owner in Palo Alto is building a new website. He has contracted with an Indian web developer, Gopesh, who lives in Bangalore India. Gopesh has agreed to be paid in bitcoin. This story will examine the use of bitcoin for outsourcing, contract services and international wire transfers.

Charitable Donations

Eugenia is the director of a children’s charity in the Philippines. Recently she has discovered bitcoin and wants to use it to reach a whole new group of foreign and domestic donors to fundraise for her charity. She’s also investigating ways to use bitcoin to distribute funds quickly to areas of need. This story will show the use of bitcoin for global fundraising across currencies and borders and the use of an open ledger for transparency in charitable organizations.

Remittances and Reverse Remittances

Gopesh, the Indian web developer, is supporting his daughter Radhika who is a student in Essex, England. Gopesh is now considering sending Radhika bitcoin, eliminating the fees he used to pay for remittances. This story will demonstrate the use of local exchange and peer-to-peer exchanges for international remittances with bitcoin.

Import/Export

Mohammed is an electronics importer in Dubai. He’s trying to use bitcoin to buy electronics from the USA and China for import into the U.A.E., to accelerate the process of payments for imports. This story will show how bitcoin can be used for large business-to-business international payments tied to physical goods.

Mining for Bitcoin

Jing is a computer engineering student in Shanghai. He has built a “mining” rig to mine for bitcoins, using his engineering skills to supplement his income. This story will examine the “industrial” base of bitcoin, the specialized equipment used to secure the bitcoin network and issue new currency.

Peer Lending

Zenab is a shopkeeper in Kisumu, Kenya and needs a loan to buy new inventory for her shop. With the assistance of a micro-lending organization, she is financing a micro-loan in bitcoin from individual lenders all across the world. This story will demonstrate the potential for bitcoin to offer peer-to-peer micro-lending by aggregating small investments, matching them with borrowers in developing nations.

Each of the stories above is based on real people and real industries that are currently using bitcoin to create new markets, new industries and innovative solutions to global economic issues.

Getting Started

To join the bitcoin network and start using the currency, all a user has to do is download an application. Since bitcoin is a standard, there are many implementations of the bitcoin client software. There is also a “reference implementation”, also known as the Satoshi Client, which is managed as an open source project by a team of developers and is derived from the original implementation written by Satoshi Nakamoto.

The three primary forms of bitcoin clients are:

Full Client

A full client, or “full node” is a client that stores the entire history of bitcoin transactions, manages the user’s wallets and can initiate transactions directly on the bitcoin network. This is similar to a standalone email server, in that it handles all aspects of the protocol without relying on any other servers or third party services.

Light Client

A lightweight client stores the user’s wallet but relies on third-party owned servers for access to the bitcoin transactions and network. The light client does not store a full copy of all transactions and therefore must trust the third party servers for transaction validation. This is similar to a standalone email client that connects to a mail server for access to a mailbox, in that it relies on a third party for interactions with the network.

Web Client

Web-clients are accessed through a web browser and store the user’s wallet on a server owned by a third party. This is similar to webmail, in that it relies entirely on a third party server.

MOBILE BITCOIN

Mobile clients, for smartphones such as those based on the Android system can either operate as full clients, light clients or web clients. Some mobile clients are synchronized with a web or desktop client, providing a multi-platform wallet across multiple devices but with a common source of funds. See (to come)

The choice of bitcoin client depends on how much control the user wants over funds. A full client will offer the highest level of control and independence for the user, but in turn put the burden of backups and security on the user. On the other end of the range of choices, a web client is the easiest to setup and use, but the security and control is shared by the user and the owner of the web service, which introduces counterparty risk. If a web-wallet service is compromised, as many have been, the users can lose all their funds. Conversely, if a user has a full client without adequate backups, they may lose their funds through a computer mishap.

For the purposes of this book, we will be demonstrating the use of a variety of bitcoin clients, from the reference implementation (the Satoshi client) to web-wallets. Some of the examples will require the use of the reference client, which exposes APIs to the wallet, network and transaction services. If you are planning to explore the programmatic interfaces into the bitcoin system, you will need the reference client.

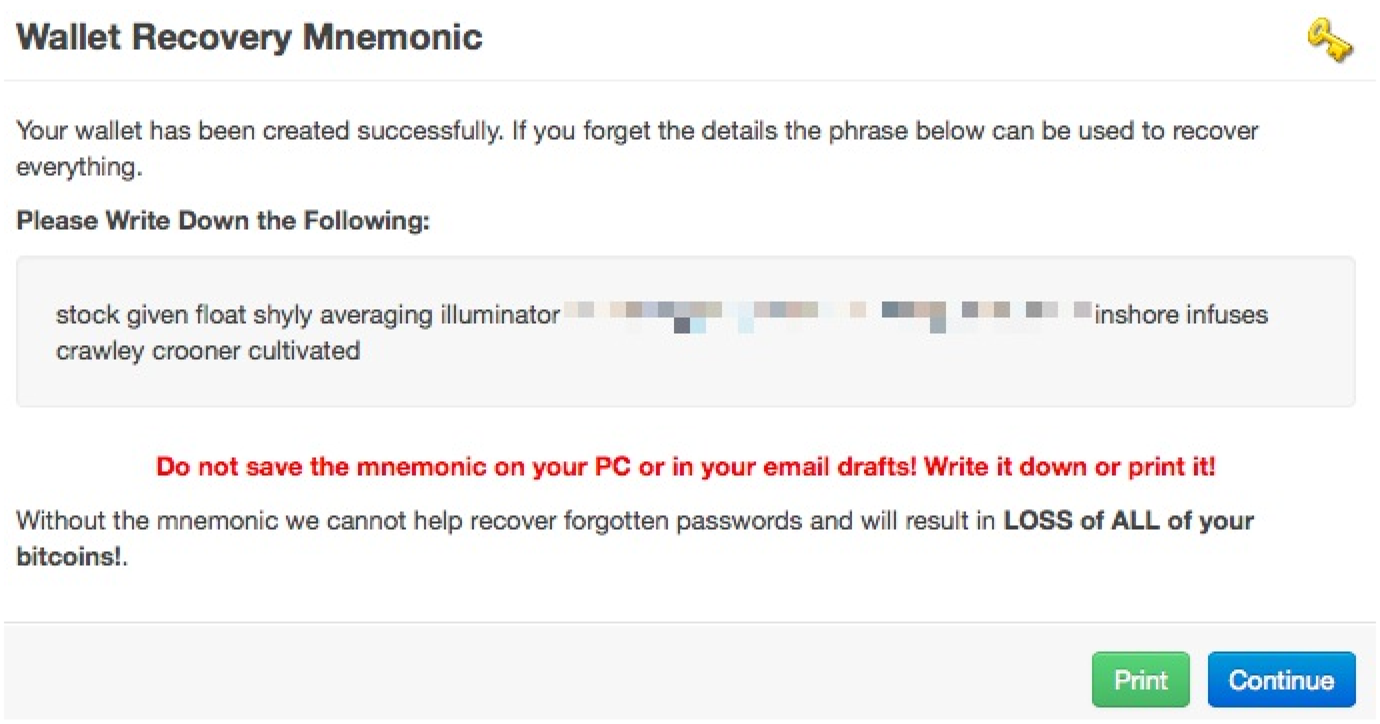
Quick Start - Web Wallet

A web-wallet is the easiest way to start using bitcoin, and is the choice of Alice who we introduced in Bitcoin Uses, Users and Their Stories. Alice is not a technical user and only recently heard about bitcoin from a friend. She starts her journey by visiting the official website bitcoin.org, where she finds a broad selection of bitcoin clients. Following the advice on the bitcoin.org site, she chooses a web-wallet by blockchain.info a popular hosted-wallet service. Following a link from bitcoin.org, she opens the blockchain.info wallet page at https://blockchain.info/wallet and selects “Start a New Wallet”. To register her new wallet, she must enter an email address, a password and prove that she is a human by completing a CAPTCHA test.

WARNING

When creating a bitcoin wallet you will need to provide a password or passphrase to protect your wallet. There are many bad actors attempting to break weak passwords, so take care to select one that cannot be easily broken. Use a combination of upper and lower-case characters, numbers and symbols. Avoid personal information such as birthdates or names of sports teams. Avoid any words commonly found in dictionaries, in any language. If you can, use a password generator to create a completely random password, at least 12 characters in length. Remember: bitcoin is money and can be instantly moved anywhere in the world - that makes it easy to steal and disappear.

Once Alice has completed the registration form, she is presented with a Wallet Recovery Mnemonic. This is a series of words that can be used to reconstruct her wallet in case she loses the password or account details. Following the instructions on screen, Alice copies the words onto paper, locking it away in a secure location.



Wallet Recovery Mnemonic

Your wallet has been created successfully. If you forget the details the phrase below can be used to recover everything.

Please Write Down the Following:

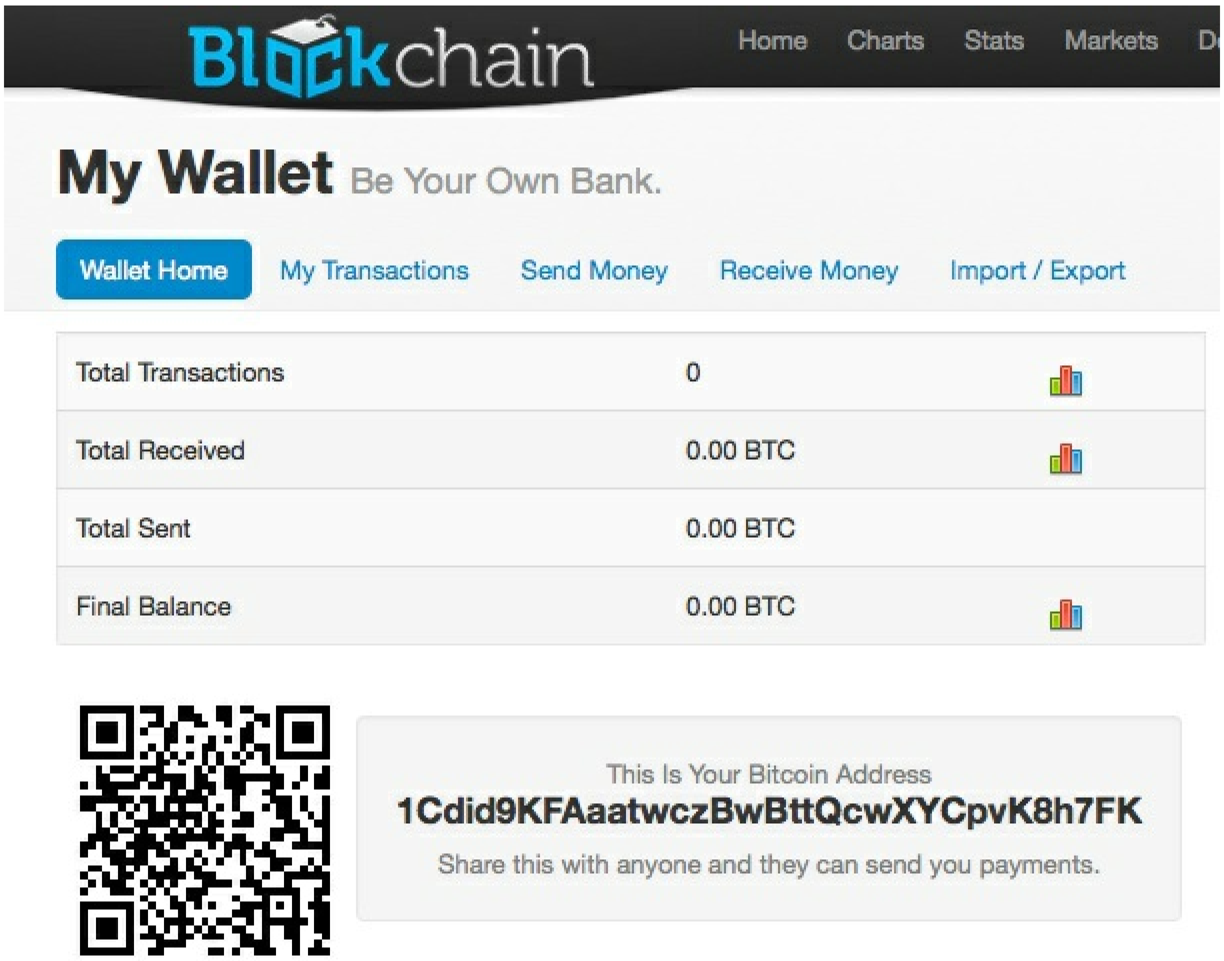
stock given float shyly averaging illuminator [redacted] inshore infuses crawley crooner cultivated

Do not save the mnemonic on your PC or in your email drafts! Write it down or print it!

Without the mnemonic we cannot help recover forgotten passwords and will result in **LOSS of ALL of your bitcoins!**.

Figure 1-1. Blockchain.info - Wallet Recovery Mnemonic

A few seconds later, Alice can start using her new bitcoin web-wallet by logging in with her account ID and password. In her web browser, she sees the web-wallet home screen:



My Wallet Be Your Own Bank.

Total Transactions	0
Total Received	0.00 BTC
Total Sent	0.00 BTC
Final Balance	0.00 BTC

This Is Your Bitcoin Address

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK

Share this with anyone and they can send you payments.

Figure 1-2. Blockchain.info - Wallet Home Screen

The most important part of this screen is Alice's *bitcoin address*. Like an email address, Alice can share this address and anyone can use it to send money directly to her new web-wallet. On the screen it appears as a long string of letters and numbers: 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Next to the wallet's bitcoin address, there is a QR-code, a form of barcode that contains the same information in a format that can be easily scanned by a smartphone's camera. Alice can print the QR code as a way to easily give her address to others without them having to type the long string of letters and numbers.

TIP

Bitcoin addresses start with the digit "1". Like email addresses, they can be shared with other bitcoin users who can use them to send bitcoin directly to your wallet. Unlike email addresses, you can create new addresses as often as you like, all of which will direct funds to your wallet. A wallet is simply a collection of addresses and the keys that unlock the funds within. There is practically no limit to the number of addresses a user can create.

Alice is now ready to start using her new bitcoin web-wallet.

Getting your first bitcoins

It is not possible to buy bitcoins at a bank, or foreign exchange kioks, at this time. It is not possible to use a credit card to buy bitcoins, either. At the time this book is being written, in 2013, it is still quite difficult to acquire bitcoins in most countries. There are a number of specialized currency exchanges where you can buy and sell bitcoin in exchange for a local currency. These operate as web-based currency markets and include:

- Bitstamp (bitstamp.net), a European currency market that supports several currencies including euros (EUR) and US dollars (USD) via wire transfer
- Coinbase (coinbase.com), a US-based currency market, based in California, that supports US dollar exchange to and from bitcoin. Coinbase can connect to US checking accounts via the ACH system

Crypto-currency exchanges such as these operate at the intersection of national currencies and crypto-currencies. As such, they are subject to national and international regulations and are often specific to a single country or economic area and specialize in the national currencies of that area. Your choice of currency exchange will be specific to the national currency you use and limited to the exchanges that operate within the legal jurisdiction of your country. It takes several days or weeks to setup the necessary accounts with the above services, as they require various forms of identification to comply with KYC (Know Your Customer) and AML (Anti-Money Laundering) banking regulations, essentially like opening a new bank account. Once you have an account on a bitcoin exchange, you can then buy or sell bitcoins quickly, just like buying a foreign currency with a brokerage account.

A more complete list can be found at http://bitcoincharts.com/markets/, a site that offers price quotes and other market data across many dozens of currency exchanges.

There are three other methods for getting bitcoins as a new user:

- Find a friend who has bitcoins and buy some from them directly. Many bitcoin users started this way.
- Use a classified service like localbitcoins.com to find a seller in your area to buy bitcoins for cash in an in-person transaction.
- Sell a product or service for bitcoin. If you're a programmer, sell your programming skills. If you have an online store, see (to come) to sell in bitcoin.

Alice was introduced to bitcoin by a friend and so she has an easy way of getting her first bitcoin while she waits for her account on a California currency market to be verified and activated.

Sending and receiving bitcoins

Alice has created her bitcoin web-wallet and she is now ready to receive funds. Her web-wallet application generated a bitcoin address and the corresponding key (an elliptic curve private key, describe in more detail in (to come)). At this point, her bitcoin address is not known to the bitcoin network or "registered" with any part of the bitcoin system. Her bitcoin address is simply a number that corresponds to a key that she can use to control access to the funds. There is no account or association between that address and an account. Until the moment this address is referenced as the recipient of value in a transaction posted on the bitcoin ledger (the blockchain), it is simply part of the vast number of possible addresses that are "valid" in bitcoin. Once it has been associated with a transaction, it becomes part of the known addresses in the network and anyone can check its balance on the public ledger.

Alice meets her friend Joe who introduced her to bitcoin at a local restaurant so they can exchange some US dollars and put some bitcoins into her account. She has brought a print out of her address and the QR code as shown on the home page of her web-wallet. There is nothing sensitive, from a security perspective, about the bitcoin address, it can be posted anywhere without risking the security of her account and it can be changed by creating a new address at any time. Alice wants to convert just $10 US dollars into bitcoin, so as not to risk too much money on this new technology. She gives Joe a $10 bill and the printout of her address so that Joe can send her the equivalent amount of bitcoin.

First, Joe has to figure out the exchange rate so that he can give the correct amount of bitcoin to Alice. There are hundreds of applications and web sites that can provide the current market rate, here are some of the most popular:

- bitcoincharts.com, a market data listing service that shows the market rate of bitcoin across many exchanges around the globe, denominated in different local currencies

- bitcoinaverage.com, a site that provides a simple view of the volume-weighted-average for each currency.
- ZeroBlock, a free Android and iOS application that can display a bitcoin price from different exchanges.

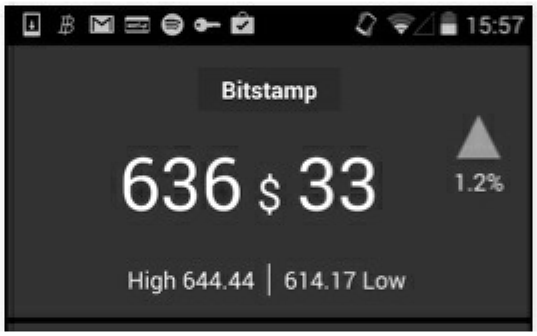
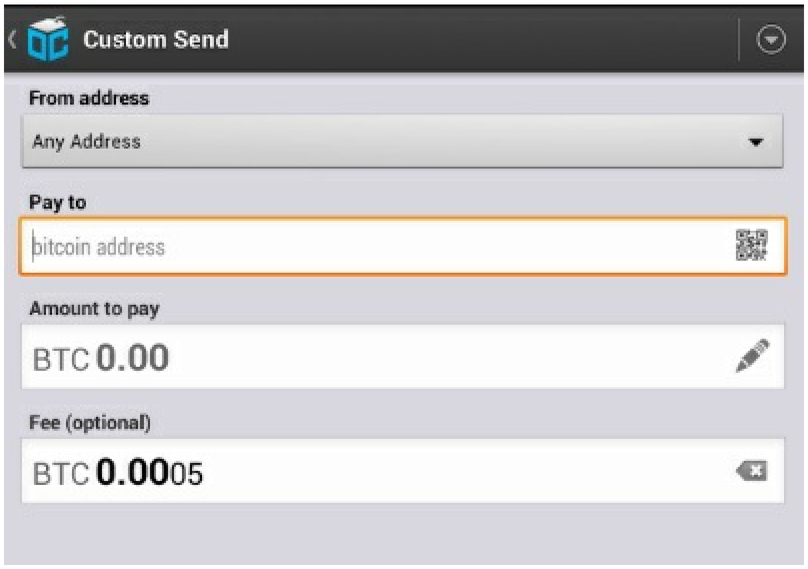


Figure 1-3. ZeroBlock - A bitcoin market-rate application for Android and iOS

Using one of the applications or websites above, Joe determines the price of bitcoin to be approximately $100 US dollars per bitcoin. At that rate, he should give Alice 0.10 bitcoin, also known as 100 milli-bits, in return for the $10 US dollars she gave him.

Once Joe has established a fair exchange price, he opens his mobile wallet application and selects to "send" bitcoin. He is presented with a screen requesting two inputs:

- The destination bitcoin address for the transaction
- The amount of bitcoin to send



From address
Any Address
Pay to
bitcoin address
Amount to pay
BTC 0.00
Fee (optional)
BTC 0.0005

Figure 1-4. Bitcoin mobile wallet - Send bitcoin screen

In the input field for the bitcoin address, there is a small icon that looks like a QR code. This allows

Joe to scan the barcode with his smartphone camera so that he doesn't have to type in Alice's bitcoin address (1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK), which is quite long and difficult to type. Joe taps on the QR code icon and activates the smartphone camera, scanning the QR code from Alice's wallet, from the printed page she brought with her. The mobile wallet application fills in the bitcoin address and Joe can check that it scanned correctly by comparing a few digits from the address with the address printed by Alice.

Joe then enters the bitcoin value for the transaction, 0.10 bitcoin. He carefully checks to make sure he has entered the correct amount, as he is about to transmit money and any mistake could be costly. Finally, he presses "Send" to transmit the transaction. Joe's mobile bitcoin wallet constructs a transaction that assigns 0.10 bitcoin to the address provided by Alice, sourcing the funds from Joe's wallet and signing the transaction with Joe's private keys. This tells the bitcoin network that Joe has authorized a transfer of value from one of his addresses to Alice's new address. As the transaction is transmitted via the peer-to-peer protocol, it quickly propagates across the bitcoin network. In less than a second, most of the well-connected nodes in the network receive the transaction and see Alice's address for the first time.

If Alice has a smartphone or laptop with her, she will also be able to see the transaction. The bitcoin ledger - a constantly growing file that records every bitcoin transaction that has ever occurred - is public, meaning that all she has to do is look up her own address and see if any funds have been sent to it. She can do this quite easily at the blockchain.info website by entering her address in the search box. The website will show her a page (https://blockchain.info/address/1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK) listing all the transactions to and from that address. If Alice is watching that page, soon after Joe hits "Send", it will update to show a new transaction transferring 0.10 bitcoin to her balance.

CONFIRMATIONS

At first, Alice's address will show the transaction from Joe as "Unconfirmed". This means that the transaction has been propagated to the network but has not yet been included in the bitcoin transaction ledger, known as the blockchain. To be included, the transaction must be "picked up" by a miner and included in a block of transactions. Once a miner has discovered a solution to the Proof-of-Work algorithm for this block, in approximately 10 minutes, the transactions within the block will be accepted as "confirmed" by the network and can be spent. The transaction is seen by all instantly, but is only "trusted" by all when it is included in a newly mined block. The more blocks mined after that block, the more trusted it is, as more and more computation is "piled" on top of it.

Alice is now the proud owner of 0.10 bitcoin which she can spend. In the next chapter we will look at her first purchase with bitcoin and examine the underlying transaction and propagation technologies in more detail.

Chapter 2. How Bitcoin Works

Transactions, Blocks, Mining and the Blockchain

The bitcoin system, unlike traditional banking and payment systems, is based on de-centralized trust. Instead of a central trusted authority, in bitcoin, trust is achieved as an emergent property from the interactions of different participants in the bitcoin system. In this chapter we will examine bitcoin from a high-level by tracking a single transaction through the bitcoin system and watch as it becomes “trusted” and accepted by the bitcoin mechanism of distributed consensus and is finally recorded on the blockchain, the distributed ledger of all transactions.

Each example below is based upon an actual transaction made on the bitcoin network, simulating the interactions between the users (Joe, Alice and Bob) by sending funds from one wallet to another. While tracking a transaction through the bitcoin network and blockchain, we will use a *blockchain explorer* site to visualize each step. A blockchain explorer is a web application that operates as a bitcoin search engine, in that it allows you to search for addresses, transactions and blocks and see the relationships and flows between them.

Popular blockchain explorers include:

- blockchain.info
- blockexplorer.com
- biteasy.com

Each of these has a search function that can take an address, transaction hash or block number and find the equivalent data on the bitcoin network and blockchain. With each example, we will provide a URL that takes you directly to the relevant entry, so you can study it in detail.

Bitcoin Overview

In the overview diagram below, we see that the bitcoin system consists of users with wallets containing keys, transactions which are propagated across the network and miners who produce (through competitive computation) the consensus blockchain, the authoritative ledger of all transactions. In this chapter, we will trace a single transaction as it travels across the network and examine the interactions between each part of the bitcoin system, at a high level. Subsequent chapters will delve deeper into the technology behind wallets, mining and merchant systems.

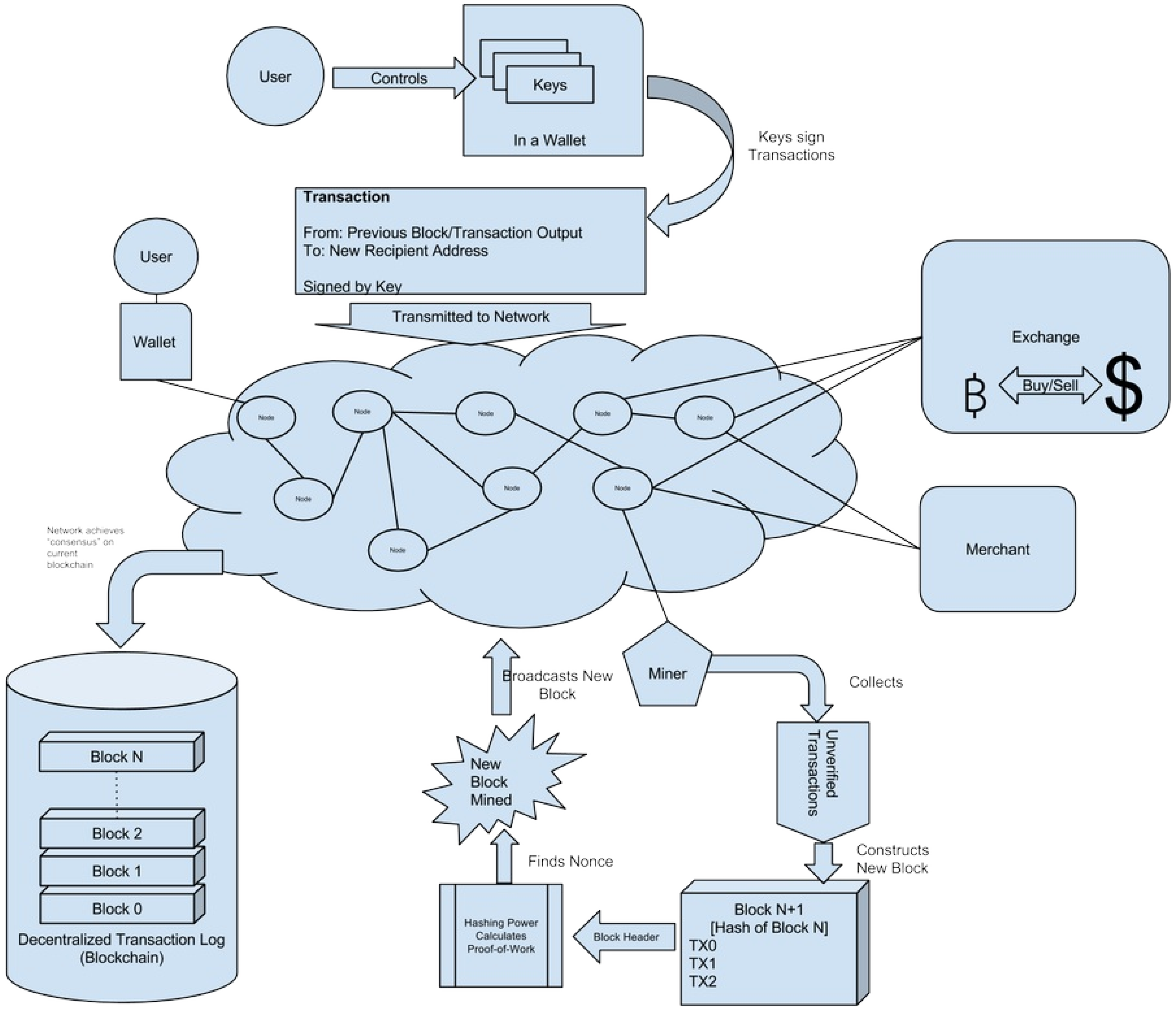


Figure 2-1. Bitcoin Overview

Buying a cup of coffee

Alice, who we introduced in the previous chapter, is a new user who has just acquired her first bitcoin. In Getting your first bitcoins, Alice met with her frined Joe to exchange some cash for bitcoin. The transaction created by Joe, funded Alice's wallet with 0.10 BTC. Now Alice will make her first retail transaction, buying a cup of coffee at Bob's coffee shop in Palo Alto, California. Bob's coffee shop recently started accepting bitcoin payments, by adding a bitcoin option to his point-of-sale system (see (to come) for information on using bitcoin for merchants/retail). The prices at Bob's Cafe are listed in the local currency (US dollars) but at the register, customers have the option of paying in either dollars or bitcoin. Alice places her order for a cup of coffee and Bob enters the transaction at the register. The point-of-sale system will convert the total price from US dollars to bitcoins at the prevailing market rate and displays the prices in both currencies, as well as showing a QR code containing a *payment request* for this transaction:

Displayed on Bob's cash register.

Total:
$1.50 USD
0.015 BTC

Figure 2-2. Payment Request QR Code - Hint: Try to scan this!

The payment request QR code above encodes the following URL, defined in BIP0021.

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

Components of the URL

A bitcoin address: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
The payment amount: "0.015"
A label for the recipient address: "Bob's Cafe"
A description for the payement: "Purchase at Bob's Cafe"
```

TIP

Unlike a QR code that simply contains a destination bitcoin address, a “payment request” is a QR encoded URL that contains a destination address, a payment amount and a generic description such as “Bob’s Cafe”. This allows a bitcoin wallet application to pre-fill the information to send the payment while showing a human-readable description to the user. See (to come), for more details. You can scan the QR code above with a bitcoin wallet application to see what Alice would see.

Bob says “That’s one-dollar-fifty, or fifteen milibits”.

Alice uses her smartphone to scan the barcode on display. Her smartphone shows a payment of 0.0150 BTC to Bob’s Cafe and she selects Send to authorize the payment. Within a few seconds (about the same time as a credit card authorization), Bob would see the transaction on the register, completing the transaction.

In the following sections we will examine this transaction in more detail, see how Alice’s wallet constructed it, how it was propagated across the network, how it was verified and finally how Bob, the owner of the cafe, can spend that amount in subsequent transactions

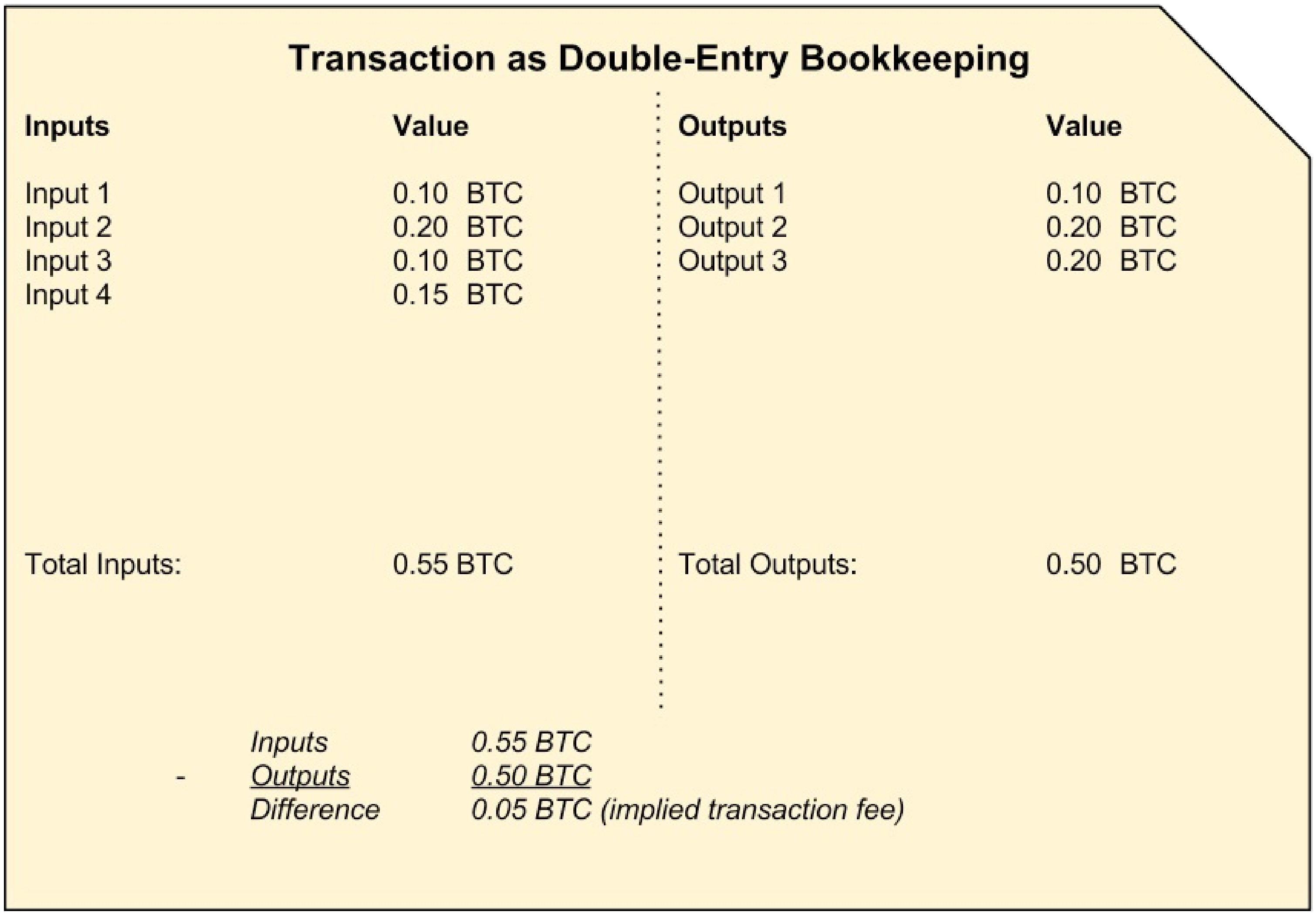
NOTE

The bitcoin network can transact in fractional values, e.g. from millibitcoins (1/1000th of a bitcoin) down to 1/100,000,000th of a bitcoin, which is known as a Satoshi. Throughout this book we'll use the term "bitcoins" to refer to any quantity of bitcoin currency, from the smallest unit (1 Satoshi) to the total number (21,000,000) of all bitcoins that will ever be mined.

Bitcoin Transactions

In simple terms, a transaction tells the network that the owner of a number bitcoins has authorized the transfer of some of those bitcoins to another owner. The new owner can now spend these bitcoins by creating another transaction that authorizes transfer to another owner, and so on, in a chain of ownership.

Transactions are like lines in a double-entry bookkeeping ledger. In simple terms, each transaction contains one or more "inputs", which are debits against a bitcoin account. On the other side of the transaction, there are one or more "outputs", which are credits added to a bitcoin account. The inputs and outputs (debits and credits) do not necessarily add up to the same amount. Instead, outputs add up to slightly less than inputs and the difference represents an implied "transaction fee", a small payment collected by the miner who includes the transaction in the ledger.



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		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC

	Inputs	*0.55 BTC*
-	*Outputs*	*0.50 BTC*
	Difference	*0.05 BTC (implied transaction fee)*

Figure 2-3. Transaction As Double-Entry Bookkeeping

The transaction contains proof of ownership for each amount of bitcoin (inputs) whose value is transfered, in the form of a digital signature from the owner, that can be independently validated by anyone. In bitcoin terms, “spending” is signing the value of a previous transaction for which you have the keys, over to a new owner identified by a bitcoin address.

TIP

Transactions move value **from** *transaction inputs* **to** *transaction outputs*. An input is where the coin value is coming from, usually a previous transaction’s output. A transaction output assigns a new owner to the value by associating it with a key. The destination key is called an *encumberance*, it imposes a requirement for a signature for the funds to be redeemed in future transactions. Outputs from one transaction can be used as inputs in a new transaction, thus creating a chain of ownership as the value is moved from address to address.

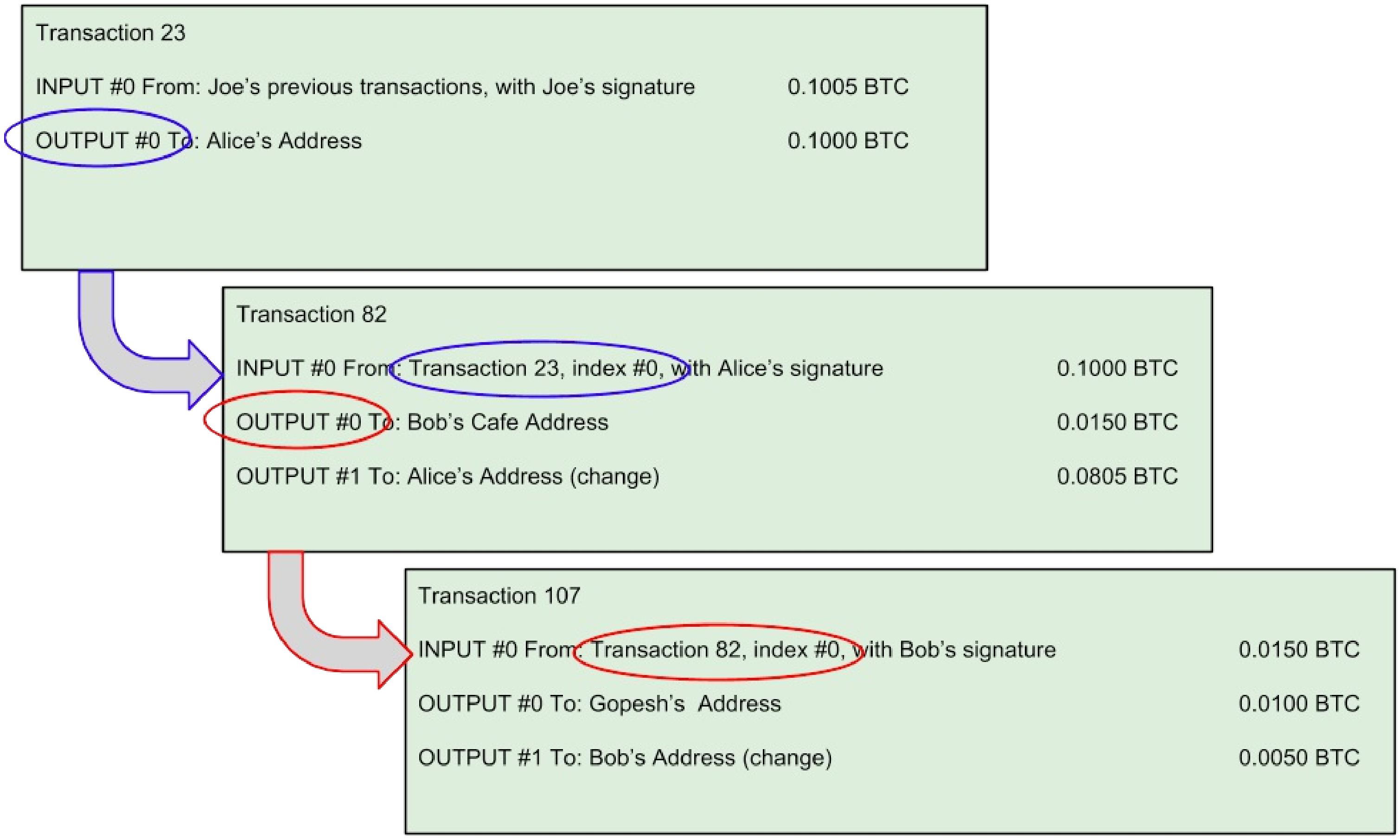

Transaction 23
INPUT #0 From: Joe's previous transactions, with Joe's signature 0.1005 BTC
OUTPUT #0 To: Alice's Address 0.1000 BTC
Transaction 82
INPUT #0 From: Transaction 23, index #0, with Alice's signature 0.1000 BTC
OUTPUT #0 To: Bob's Cafe Address 0.0150 BTC
OUTPUT #1 To: Alice's Address (change) 0.0805 BTC
Transaction 107
INPUT #0 From: Transaction 82, index #0, with Bob's signature 0.0150 BTC
OUTPUT #0 To: Gopesh's Address 0.0100 BTC
OUTPUT #1 To: Bob's Address (change) 0.0050 BTC

Figure 2-4. Transaction Chain

Alice's payment to Bob's Cafe utilizes a previous transaction as its input. In the previous chapter Alice received bitcoin from her friend Joe in return for cash. That transaction has a number of bitcoins locked (encumbered) against Alice's key. Her new transaction to Bob's Cafe references the previous transaction as an input and creates new outputs to pay for the cup of coffee and receive change. The transactions form a chain, where the inputs from the latest transaction correspond to outputs from previous transactions. Alice's key provides the signature which unlocks those previous transaction outputs, thereby proving to the bitcoin network that she owns the funds. She attaches the payment for coffee to Bob's address, thereby "encumbering" that output with the requirement that Bob produces a signature in order to spend that amount. This represents a transfer of value between Alice and Bob.

Common Transaction Forms

The most common form of transaction is a simple payment from one address to another, which often includes some "change" returned to the original owner. This type of transaction has one input and two outputs and is shown below:

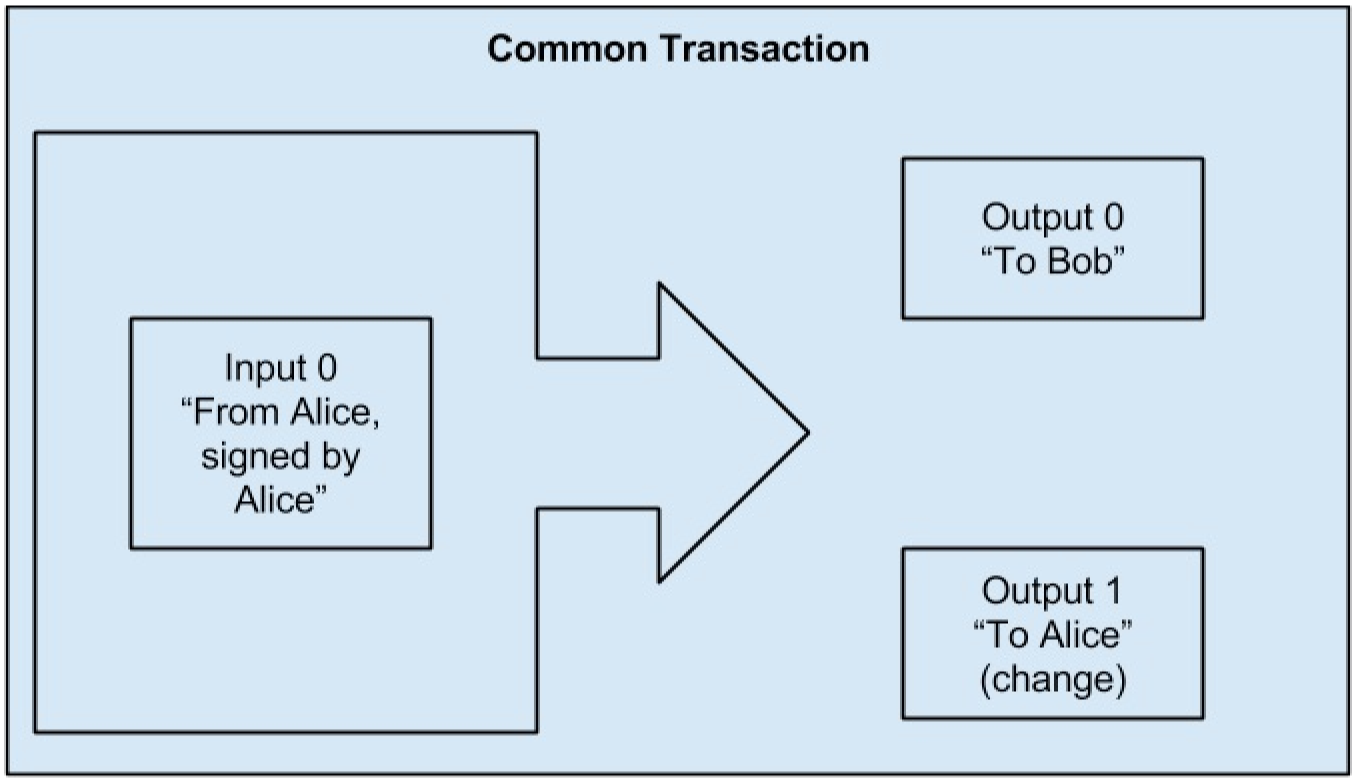


Figure 2-5. Most Common Transaction

Another common form of transaction is a transaction that aggregates several inputs into a single output. This represents the real-world equivalent of exchanging a pile of coins and currency notes for a single larger note. Transactions like these are sometimes generated by wallet applications to cleanup lots of smaller amounts that were received as change for payments.

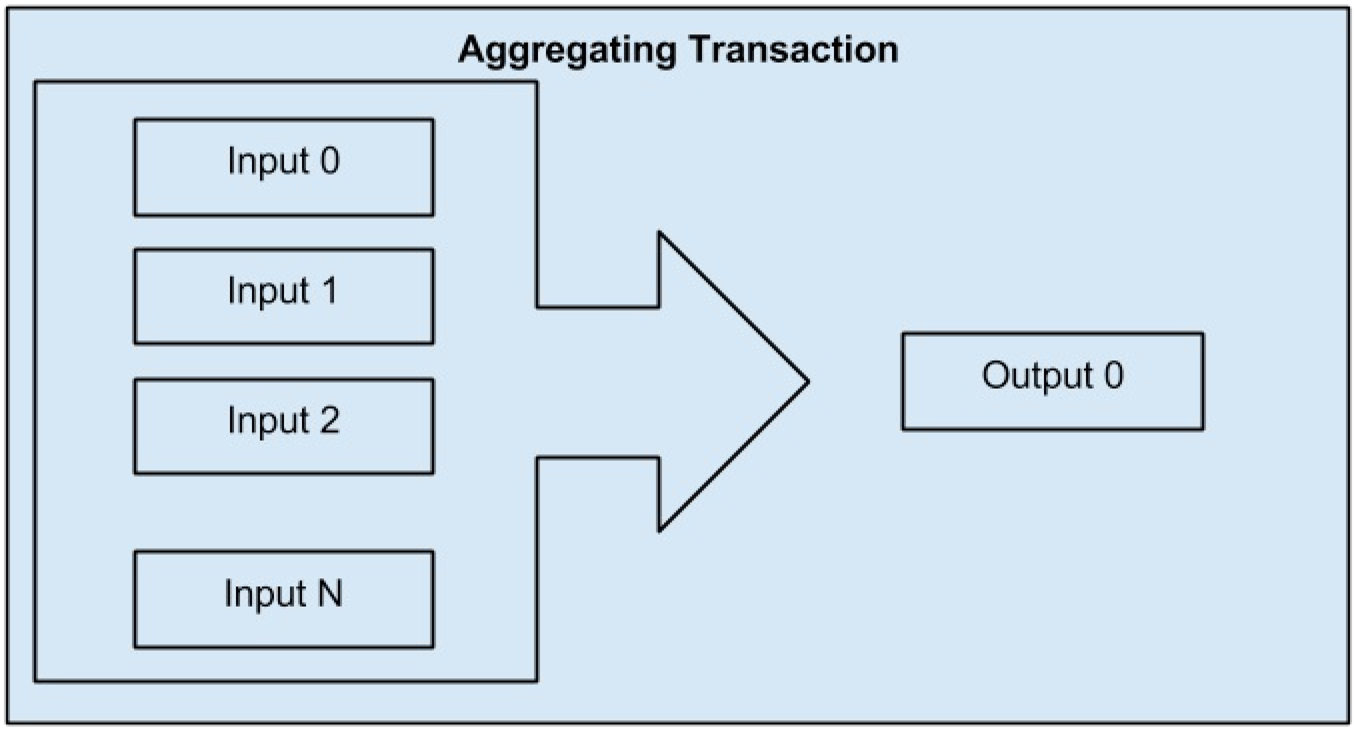


Figure 2-6. Transaction Aggregating Funds

Finally, another transaction form that is seen often on the bitcoin ledger is a transaction that distributes one input to multiple outputs representing multiple recipients. This type of transaction is sometimes used by commercial entities to distribute funds, such as when processing payroll payments to multiple employees.

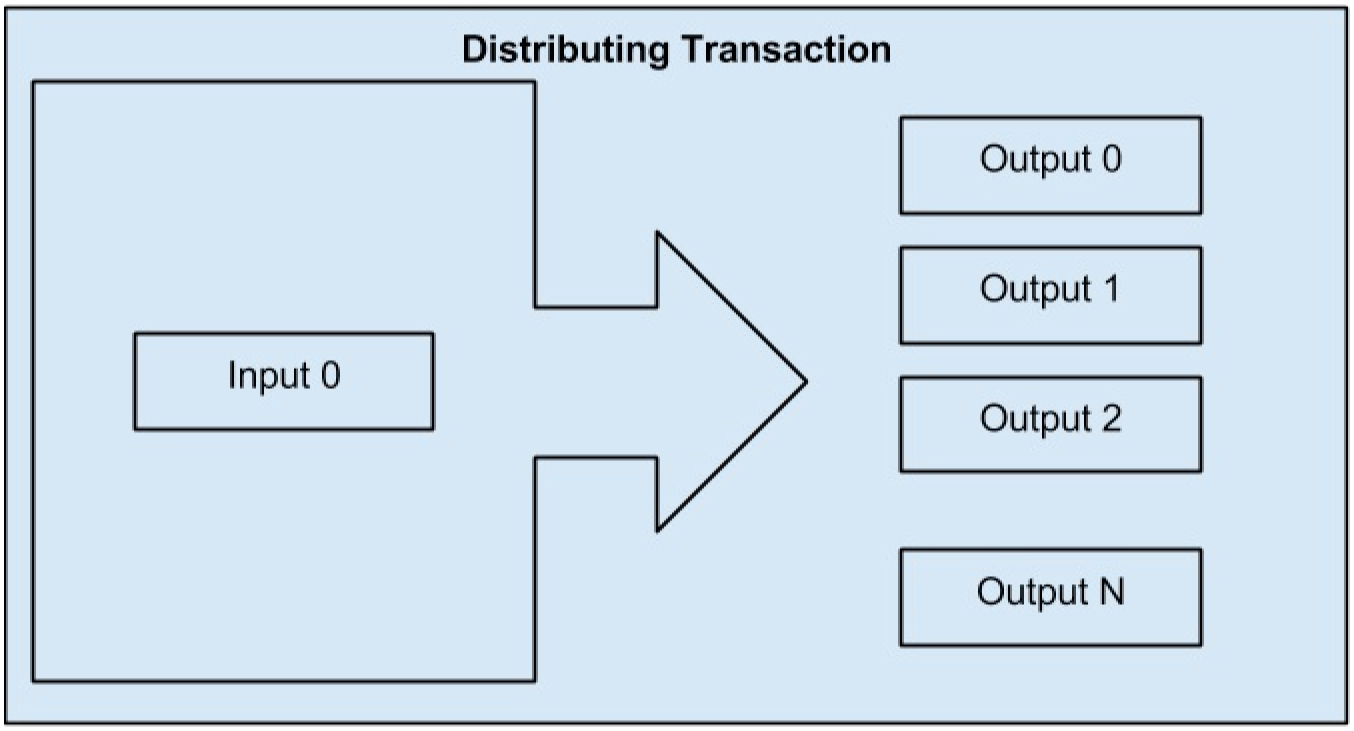


Figure 2-7. Transaction Distributing Funds

Constructing A Transaction

Alice's wallet application contains all the logic for selecting appropriate inputs and outputs to build a transaction to Alice's specification. Alice only needs to specify a destination and an amount and the rest happens in the wallet application without her seeing the details. Importantly, a wallet application can construct transactions even if completely offline. Like writing a cheque at home and later sending it to the bank in an envelope, the transaction does not need to be constructed and signed while connected to the bitcoin network, it only has to be sent to the network eventually for it to be executed.

Getting the right inputs

Alice's wallet application will first have to find inputs that can pay for the amount she wants to send to Bob. Most wallet applications keep a small database of "unspent transaction outputs" that are locked (encumbered) with the wallet's own keys. Therefore, Alice's wallet would contain a copy of the transaction output from Joe's transaction which was created in exchange for cash (see (to come)). A bitcoin wallet application that runs as a full-index client actually contains a copy of **every unspent output** from every transaction in the blockchain. This allows a wallet to construct transaction inputs as well as to quickly verify incoming transactions as having correct inputs.

If the wallet application does not maintain a copy of unspent transaction outputs, it can query the bitcoin network to retrieve this information, using a variety of APIs available by different providers, or by asking a full-index node using the bitcoin JSON RPC API. Below we see an example of a RESTful API request, constructed as a HTTP GET command to a specific URL. This URL will return all the unspent transaction outputs for an address, giving any application the information it needs to construct transaction inputs for spending. We use the simple command-line HTTP client *cURL* to retrieve the response:

Lookup all the unspent outputs for Alice's address 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK.

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

{

        "unspent_outputs":[

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

        ]
}
```

The response above shows that the bitcoin network knows of one unspent output (one that has not been redeemed yet) under the ownership of Alice's address *+1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK+*. The response includes the reference to the transaction in which this unspent output is contained (the payment from Joe) and it's value in Satoshis, at 10 million, equivalent to 0.10 bitcoin. With this information, Alice's wallet application can construct a transaction to transfer that value to new owner addresses.

TIP

Lookup the transaction from Joe to Alice, to see the information referenced above, as it is stored in the bitcoin blockchain. Using the blockchain explorer web application, follow the URL below:

https://blockchain.info/tx/7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18

As you can see, Alice's wallet contains enough bitcoins in a single unspent output to pay for the cup of coffee. Had this not been the case, Alice's wallet application might have to "rummage" through a pile of smaller unspent outputs, like picking coins from a purse, until it could find enough to pay for coffee. In both cases, there might be a need to get some change back, which we will see in the next section, as the wallet application creates the transaction outputs (payments).

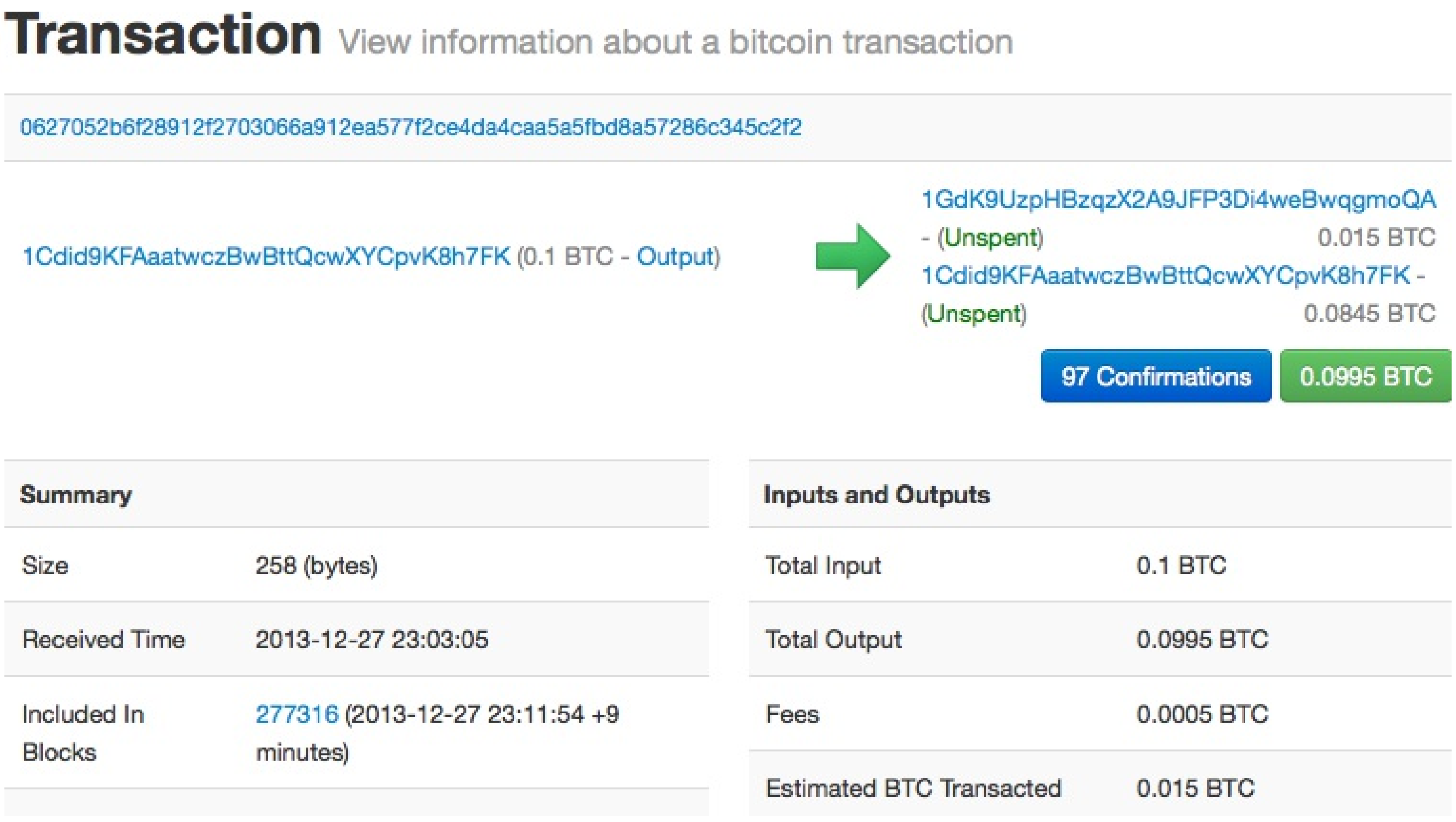
Creating the outputs

A transaction output is created in the form of a script, that creates an encumberance on the value and can only be redeemed by the introduction of a solution to the script. In simpler terms, Alice's transaction output will contain a script that says something like "This output is payable to whoever can present a signature from the key corresponding to Bob's public address". Since only Bob has the wallet with the keys corresponding to that address, only Bob's wallet can present such a signature to redeem this output. Alice will therefore "encumber" the output value with a demand for a signature from Bob.

This transaction will also include a second output, because Alice's funds are in a the form of a 0.10 BTC output, too much money for the 0.015 BTC cup of coffee. Alice will need 0.085 BTC in change. Alice's change payment is created *by Alice's wallet* in the very same transaction as the payment to Bob. Essentially, Alice's wallet breaks her funds into two payments, one to Bob, one back to herself. She can then use the change output in a subsequent transaction, thus spending it later.

Finally, for the transaction to be processed by the network in a timely fashion, Alice's wallet application will add a small fee. This is not explicit in the transaction, it is implied by the difference between inputs and outputs. If instead of taking 0.085 in change, Alice creates only 0.0845 as the second output, there will be 0.0005 BTC (half a millibitcoin) left over. The input's 0.10 BTC is not fully spent with the two outputs, as they will add up to less than 0.10. The resulting difference is the *transaction fee* which is collected by the miner as a fee for including the transaction in a block and putting it on the blockchain ledger.

The resulting transaction can be seen using a blockchain explorer web application



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	
Size	258 (bytes)
Received Time	2013-12-27 23:03:05
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)

Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Estimated BTC Transacted	0.015 BTC

Figure 2-8. Alice's transaction to Bob's Cafe

Use the following link to see it the transaction on the bitcoin blockchain:

Link to Alice's transaction on the bitcoin blockchain.

https://blockchain.info/tx/0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

Adding the transaction to the ledger

The transaction created by Alice's wallet application is 258 bytes long and contains everything necessary to confirm ownership of the funds and assign new onwers. Now, the transaction must be transmitted to the bitcoin network where it will become part of the distributed ledger, the blockchain. In the next section we will see how a transaction becomes part of a new block and how the block is "mined". Finally, we will see how the new block, once added to the blockchain is increasingly trusted by the network as more blocks are added.

Transmitting the transaction

Since the transaction contains all the information necessary to process, it does not matter how or where it is transmitted to the bitcoin network. The bitcoin network is a peer-to-peer network, with each bitcoin client participating by connecting to several other bitcoin clients. The purpose of the bitcoin network is to propagate transactions and blocks to all participants.

How it propagates

Alice's wallet application can send the new transaction to any of the other bitcoin clients it is connected to, over WiFi or mobile data, or any Internet connection. Her bitcoin wallet does not have to be connected to Bob's bitcoin wallet directly and she does not have to use the Internet connection offered by the cafe, though both those options are possible too. Any bitcoin network node (other client) that receives a valid transaction it has not seen before, will immediately forward it to other nodes it is connected to. Thus, the transaction rapidly propagates out across the peer-to-peer network, reaching a large percentage of the nodes within a few seconds.

Bob's view

If Bob's bitcoin wallet application is directly connected to Alice's wallet application, it may be the first node to receive the transaction. However, even if Alice's wallet sends it through other nodes, the transaction will reach Bob's wallet within a few seconds. Bob's wallet will immediately identify Alice's transaction as an incoming payment because it contains outputs redeemable by Bob's keys. Bob's wallet application can also independently verify that the transaction is well-formed, uses previously-unspent inputs and contains sufficient transaction fees to be included in the next block. At this point, Bob can assume, with little risk, that the transaction will shortly be included in a block and confirmed.

TIP

A common misconception about bitcoin transactions is that they must be "confirmed" by waiting 10 minutes for a new block, or up to sixty minutes for a full six confirmations. While confirmations ensure the transaction has been accepted by the whole network, for small value items like a cup of coffee, such a delay is unecessary. A merchant may accept a valid small-value transaction with no confirmations, with no more risk than a credit card payment made without ID or a signature, as many do today

Bitcoin Mining

The transaction is now propagated on the bitcoin network. It does not become part of the shared ledger (the *blockchain*) until it is verified and included in a block, in a process called *mining*. See (to come) for a detailed explanation.

The bitcoin system of trust is based on computation. Transactions are bundled into *blocks* which require an enormous amount of computation to prove, but only a small amount of computation to verify as proven, in a process called *mining*. Mining serves two purposes in bitcoin:

- Mining creates new bitcoins in each block, almost like a central bank printing new money. The amount of bitcoin created is fixed and diminishes with time

- Mining creates trust by ensuring that transactions are only confirmed if enough computational power was devoted to the block that contains them. More blocks mean more computation which means more trust.

A good way to describe mining is like a giant competitive game of sudoku that resets every time someone finds a solution and whose difficulty automatically adjusts so that it takes approximately 10 minutes to find a solution. Imagine a giant sudoku puzzle, several thousand rows and columns in size. If I show you a completed puzzle you can verify it quite quickly. If it is empty, however, it takes a lot of work to solve! The difficulty of the sudoku can be adjusted by changing its size (more or fewer rows and columns), but it can still be verified quite easily even if it is very large. The "puzzle" used in bitcoin is based on a cryptographic hash and exhibits similar characteristics: it is assymetrically hard to solve, but easy to verify and its difficulty can be adjusted.

In Bitcoin Uses, Users and Their Stories we introduced Jing, a computer engineering student in Shanghai. Jing is participating in the bitcoin network as a miner. Every 10 minutes or so, Jing joins thousands of other miners in a global race to find a solution to a block of transactions. Finding such a solution, the so-called "Proof-of-Work" requires quadrillions of hashing operations per second, across the entire bitcoin network. The algorithm for "Proof-of-Work" involves repeatedly hashing the header of the block and a random number with the SHA256 cryptographic algorithm, until a solution matching a pre-determined pattern emerges. The first miner to find such a solution wins the round of competition and publishes that block into the blockchain.

Jing started mining in 2010 using a very fast desktop computer to find a suitable Proof-of-Work for new blocks. As more miners started joining the bitcoin network, the difficulty of the problem increased rapidly. Soon, Jing and other miners upgraded to more specialized hardware, such as Graphical Processing Units (GPU), as used in gaming desktops or consoles. As this book is written, by 2014, the difficulty is so high that it is only profitable to mine with Application Specific Integrated Circuits, essentially hundreds of mining algorithms printed in hardware, running in parallel on a single silicone chip. Jing also joined a "mining pool", which much like a lottery-pool allows several participants to share their efforts and the rewards. Jing now runs two ASIC machines, which are USB connected devices, to mine for bitcoin 24 hours a day. He pays his electricity costs by selling the bitcoin he is able to generate from mining, creating some income from the profits. His computer runs a copy of bitcoind, the reference bitcoin client, as a back-end to his specialized mining software.

Mining transactions in blocks

A transaction transmitted across the network is not verified until it becomes part of the global distributed ledger, the blockchain. Every ten minutes, miners generate a new block, which contains all the transactions since the last block. New transactions are constantly flowing into the network from user wallets and other applications. As these are seen by the bitcoin network nodes, they get added to a temporary "pool" of unverified transactions maintained by each node. As miners build a new block, they add unverified transactions from this pool to a new block and then attempt to solve a very hard problem (aka Proof-of-Work) to prove the validity of that new block. The process of mining is explained in detail in (to come)

Transactions are added to the new block, prioritized by the highest-fee transactions first and a few other criteria. Each miner starts the process of mining a new block of transactions as soon as they receive the previous block from the network, knowing they have lost that previous round of competition. They immediately create a new block, fill it with transactions and the fingerprint of the previous block and start calculating a the Proof-of-Work for the new block. Each miner includes a special transaction in their block, one that pays their own bitcoin address a reward of newly created bitcoins (currently 25 BTC per block). If they find a solution that makes that block valid, they “win” this reward because their successful block is added to the global blockchain and the reward transaction they included becomes spendable. Jing, who participates in a mining pool, has setup his software to create new blocks that assign the reward to a pool address. From there, a share of the reward is distributed to Jing and other miners in proportion to the amount of work they contributed in the last round.

Alice’s transaction was picked up by the network and included in the pool of unverified transactions. Since it had sufficient fees, it was included in a new block generated by Jing’s mining pool. Approximately 5 minutes after the transaction was first transmitted by Alice’s wallet, Jing’s ASIC miner found a solution for the block and published it as block #277316, containing 419 other transactions. Jing’s ASIC miner published the new block on the bitcoin network, where other miners validated it and started the race to generate the next block.

You can see the block that includes Alice’s transaction here: https://blockchain.info/block-height/277316

A few minutes later, a new block, #277317 is mined by another miner. As this new block is based on the previous block (#277316) that contained Alice’s transaction, it added even more computation on top of that block, thereby strengthening the trust in those transactions. One block mined on top of the one containing the transaction, is called “one confirmation” for that transaction. As the blocks pile on top of each other, it becomes exponentially harder to reverse the transaction, thereby making it more and more trusted by the network.

In the diagram below, we can see block #277316, the one which contains Alice’s transaction. Below it are 277,315 blocks, linked to each other in a chain of blocks (blockchain) all the way back to block #0, the genesis block. Over time, as the “height” in blocks increases, so does the computation difficulty for each block and the chain as a whole. The blocks mined after the one that contains Alice’s transaction act as further assurance, as they pile on more computation in a longer and longer chain. The blocks above count as “confirmations”. By convention, any block with more than 6 confirmation is considered irrevocable, as it would require an immense amount of computation to invalidate and re-calculate six blocks. We will examine the process of mining and the way it builds trust in more detail in (to come).

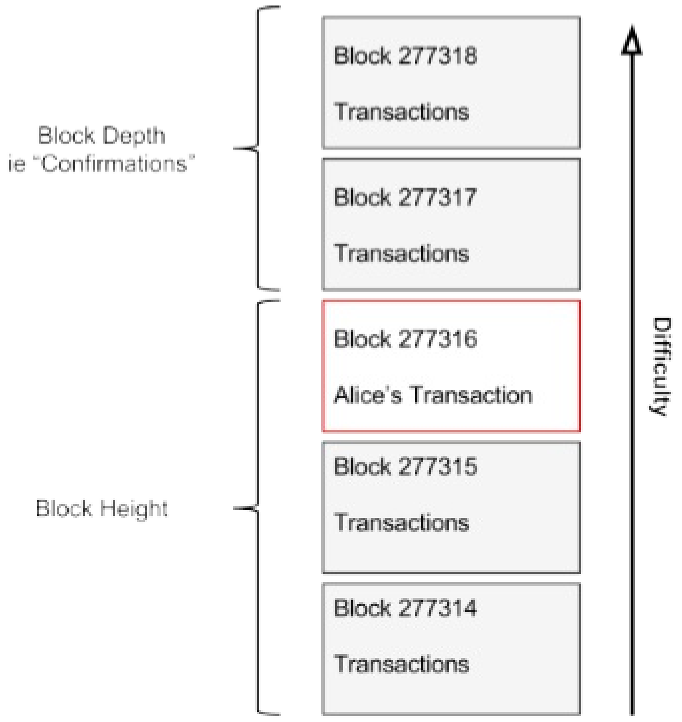


Figure 2-9. Alice's transaction included in block #277,317

Spending the transaction

Now that Alice's transaction has been embedded in the blockchain as part of a block, it is part of the distributed ledger of bitcoin and visible to all bitcoin applications. Each bitcoin client can independently verify the transaction as valid and spendable. Full-index clients can track the source of the funds from the moment the bitcoins were first generated in a block, incrementally from transaction to transaction, until they reach Bob's address. Lightweight clients can do a Simple Payment Verification (See SPV:(to come)) by confirming that the transaction is in the blockchain and has several blocks mined after it, thus providing assurance that the network accepts it as valid.

Bob can now spend the output from this and other transactions, by creating his own transactions that reference these outputs as their inputs and assign them new ownership. For example, Bob can pay a contractor or supplier by transferring value from Alice's coffee cup payment to these new owners. Most likely, Bob's bitcoin software will aggregate many small payments into a larger payment, perhaps concentrating all the day's bitcoin revenue into a single transaction. This would move the various payments into a single address, utilized as the store's general "checking" account. For a diagram of an aggregating transaction, see Figure 2-6.

As Bob spends the payments received from Alice and other customers, he exends the chain of transactions, which in turn are added to the global blochcain ledger for all to see and trust. Let's assume that Bob pays his web designer Gopesh in Bangalore for a new web site page. Now the chain of transactions will look like this:

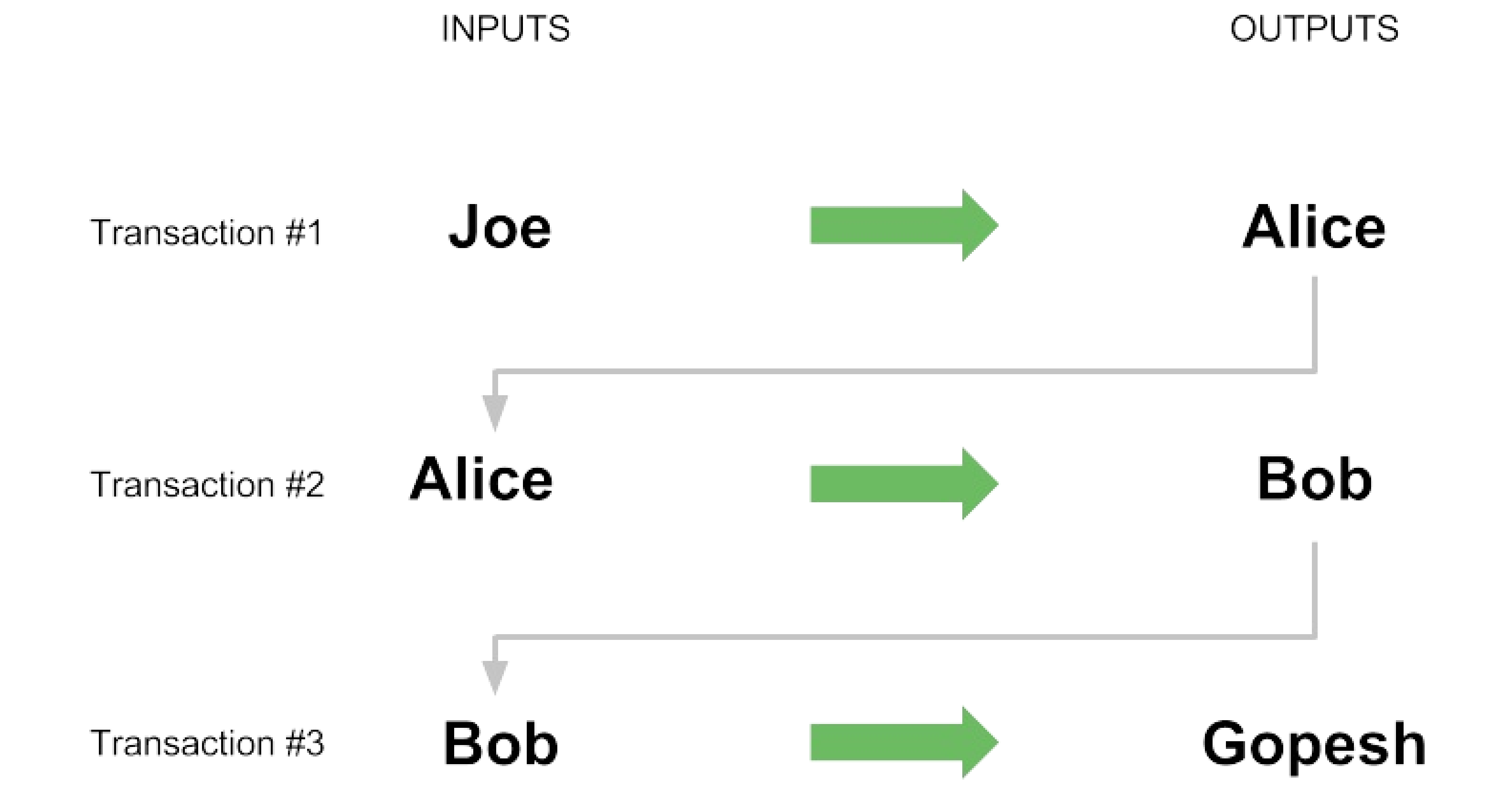


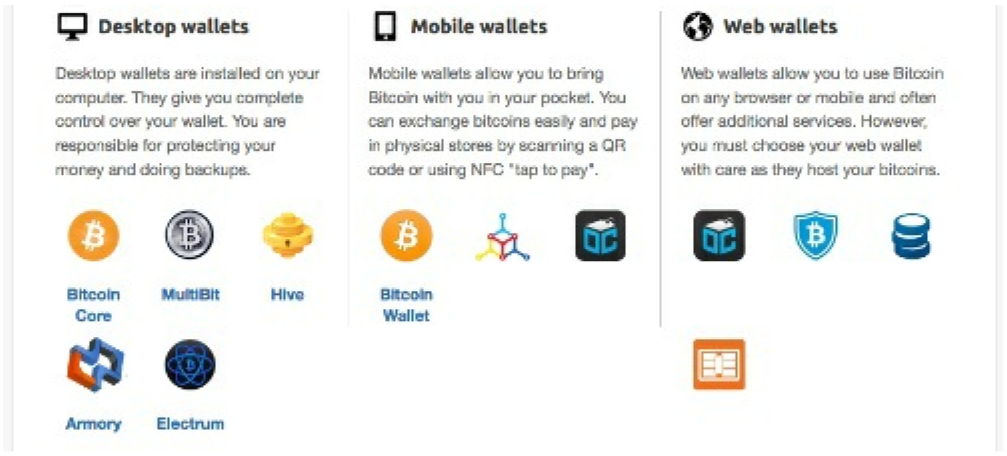
Figure 2-10. Alice's transaction as part of a transaction chain from Joe to Gopesh

Chapter 3. The Bitcoin Client

Bitcoin Core - The Reference Implementation, aka Satoshi Client

You can download the Reference Client, also known as *Bitcoin Core* from bitcoin.org. The reference client implements all aspects of the bitcoin system, including wallets, a transaction verification engine with a full copy of the entire transaciton ledger (blockchain) and a full network node in the peer-to-peer bitcoin network.

Go to http://bitcoin.org/en/choose-your-wallet and select "Bitcoin Core" to download the reference client. Depending on your operating system, you will download an executable installer. For Windows, this is either a ZIP archive or an EXE executable. For Mac OS it is DMG disk image. Linux versions include a PPA package for Ubuntu or a TAR.GZ archive.



Desktop wallets are installed on your computer. They give you complete control over your wallet. You are responsible for protecting your money and doing backups.

Mobile wallets allow you to bring Bitcoin with you in your pocket. You can exchange bitcoins easily and pay in physical stores by scanning a QR code or using NFC "tap to pay".

Web wallets allow you to use Bitcoin on any browser or mobile and often offer additional services. However, you must choose your web wallet with care as they host your bitcoins.

Figure 3-1. Bitcoin - Choose A Bitcoin Client

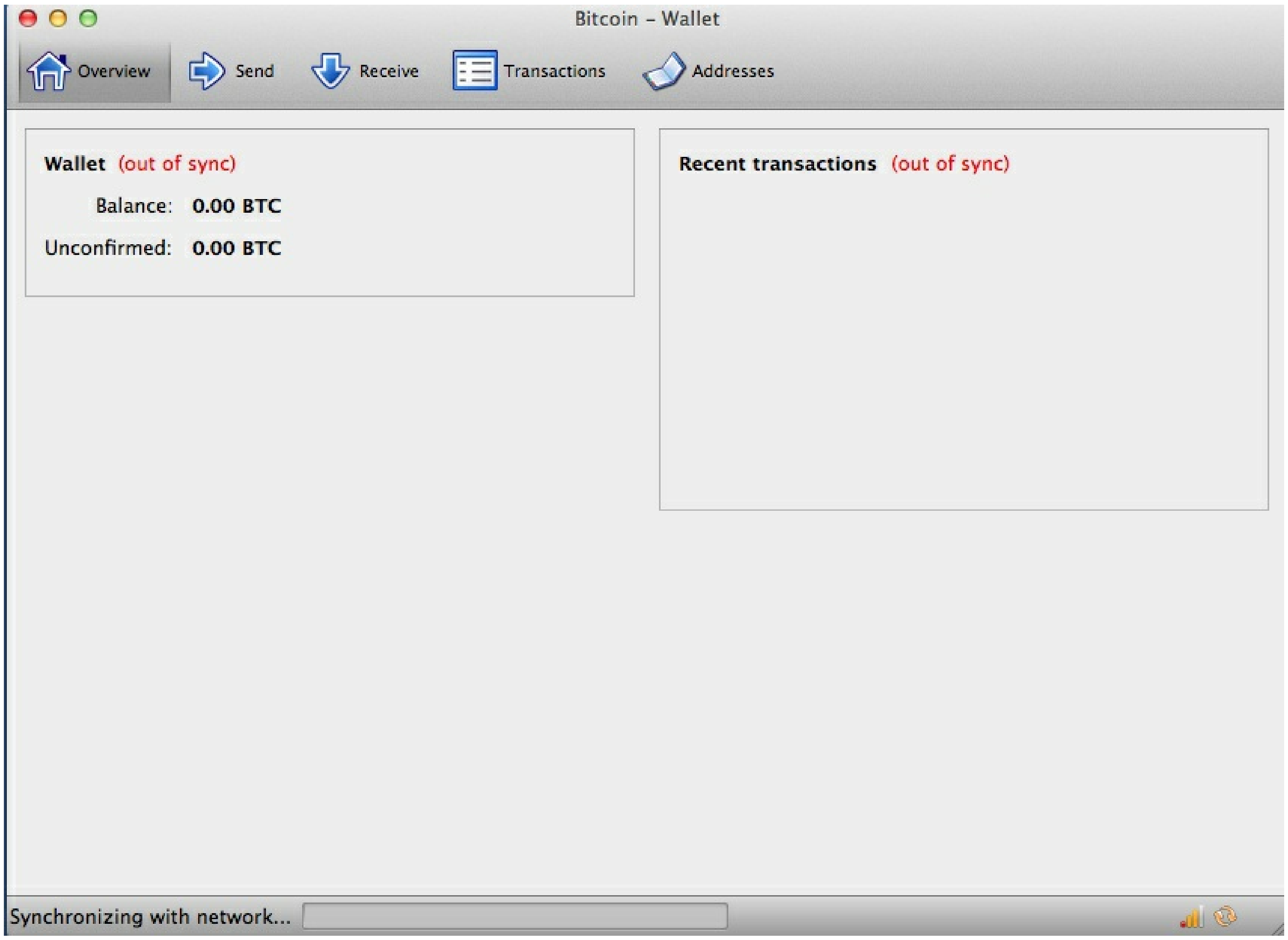
Bitcoin Core - Running the client for the first time

If you download an installable package, such as an EXE, DMG or PPA, you can install it the same way as any application on your operating system. For Windows, run the EXE and follow the step-by-step instructions. For Mac OS, launch the DMG and drag the Bitcoin-QT icon into your Applications folder. For Ubuntu, double-click on the PPA in your File Explorer and it will open the package manager to install the package. Once you have completed installation you should have a new application "Bitcoin-Qt" in your application list. Double-click on the icon to start the bitcoin client.

The first time you run Bitcoin Core it will start downloading the blockchain, a process that may take several days. Leave it running in the background, until it displays "Synchronized" and no longer shows "Out of sync" next to the balance.

TIP

Bitcoin Core keeps a full copy of the transaction ledger (blockchain), with every transaction that has ever occured on the bitcoin network since its inception in 2009. This data set is several gigabytes in size (approximately 16GB in late 2013) and is downloaded incrementally over several days. The client will not be able to process transactions or update account balances until the full blockchain dataset is downloaded. During that time, the client will display "Out of sync" next to the account balances and show "Synchronizing" in the footer. Make sure you have enough disk space, bandwidth and time to complete the initial synchronization.



Wallet (out of sync)
Balance: 0.00 BTC
Unconfirmed: 0.00 BTC

Recent transactions (out of sync)

Synchronizing with network...

Figure 3-2. Bitcoin Core - The Graphical User Interface, during the blockchain initialization

Bitcoin Core - Compiling the client from the source code

For developers, there is also the option to download the full source code, either as a ZIP archive or by cloning the authoritative source repository from Github. Go to https://github.com/bitcoin/bitcoin and select "Download ZIP" from the sidebar. Alternatively, use the git command line to create a local copy of the source code on your system. In the example below, we are cloning the source code from a unix-like command-line, in Linux or 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

The instructions and resulting output may vary from version to version. Follow the documentation that comes with the code even if it differs from the instructions you see here and don't be surprised if the output displayed on your screen is slightly different from the examples here.

When the git cloning operation has complete, you will have a complete local copy of the source code repository in the directory *bitcoin*. Change to this directory by typing cd bitcoin at the prompt:

```
$ cd bitcoin
```

By default, the local copy will be synchronized with the most recent code which may be an unstable or "beta" version of bitcoin. Before compiling the code, we want to select a specific version, by checking out a release *tag*. This will synchronize the local copy with a specific snapshot of the code repository identified by a keyword tag. Tags are used by the developers to mark specific releases of the code by version number. First, to find the available tags, we use the git tag command:

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

[... many more tags ...]

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

The list of tags shows all the released versions of bitcoin. By convention, *release candidates*, which are intended for testing, have the suffix "rc". Stable releases that can be run on production systems have no suffix. From the list above, we select the highest version release, which at this time is v0.9.0rc1. To synchronize the local code with this version, we use the git checkout command:

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

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

The source code includes documentation, which can be found in a number of files. Review the main documentation located in README.md in the bitcoin directory, by typing more README.md at the prompt, using the space bar to progress to the next page. In this chapter we will build the command-line bitcoin client, also known as bitcoind on Linux. Review the instructions for compiling the bitcoind command-line client on your platform by typing more doc/build-unix.md. Alternative instructions for Mac OSX and Windows can be found in the doc directory, as build-os.md or build-msw.md respectively.

Carefully review the build pre-requisited which are in the first part of the build documentation. These are libraries that must be present on your system before you can begin to compile bitcoin. If these pre-requisites are missing the build process will fail with an error. If this happens because you missed a pre-requisite, you can install it and then resume the build process from where you left off. Assuming the pre-requisites are installed, we start the build process by generating a set of build scripts using the autogen.sh script.

TIP

The bitcoind build process was changed to use the autogen/configure/make system starting with version 0.9. Older versions use a simple Makefile and work slightly differently from the example below. Follow the instructions for the version you want to compile. The autogen/configure/make introduced in 0.9 is likely to be the build system used for all future versions of the code and is the system demonstrated in the examples below.

```
$ ./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'
$
```

The autogen.sh script creates a set of automatic configuation scripts that will interrogate your system to discover the correct settings and ensure you have all the necessary libraries to compile the code. The most important of these is the configure script that offers a number of different options to customize the build process. Type ./configure --help to see the various options:

```
$ ./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

[... many more options and variables are displayed below ...]

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]

[... more options ...]

Use these variables to override the choices made by `configure' or to help
it to find libraries and programs with nonstandard names/locations.

Report bugs to <info@bitcoin.org>.

$
```

The configure script allows you to enable or disable certain features of bitcoind through the use of the --enable-FEATURE and --disable-FEATURE flags, where FEATURE is replaced by the feature name, as listed in the help output above. In this chapter, we will build the bitcoind client with all the default features, so we won’t be using these flags, but you should review them to understand what optional features are part of the client. Next, we run the configure script to automatically discover all the necessary libraries and create a customized build script for our system:

```
$ ./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

[... many more system features are tested ...]

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
$
```

If all goes well, the configure command will end by creating the customized build scripts that will allow us to compile bitcoind. If there are any missing libraries or errors, the configure command will terminate with an error instead of creating the build scripts as shown above. If an error occurs, it is most likely a missing or incompatible library. Review the build documentation again and make sure you install the missing pre-requisites, then run configure again and see if that fixes the error. Next, we will compile the source code, a process that can take up to an hour to complete. During the compilation process you should see output every few seconds or every few minutes, or an error if something goes wrong. The compilation process can be resumed at any time if interrupted. Type make to start compiling:

```
$ 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

[... many more compilation messages follow ...]

  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'
$
```

If all goes well, bitcoind is now compiled. The final step is to install the bitcoind executable into the system path, using the make command:

```
$ 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'
$
```

We can confirm that bitcoin is correctly installed, as follows:

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

The default installation of bitcoind puts it in /usr/local/bin. When we first run bitcoind it will remind us to create a configuration file with a strong password for the JSON-RPC interface. We run it by typing bitcoind into the terminal:

```
$ 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
(you do not need to remember this password)
The username and password MUST NOT be the same.
If the file does not exist, create it with owner-readable-only file permissions.
It is also recommended to set alertnotify so you are notified of problems;
for example: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

Edit the configuration file in your preferred editor and set the parameters, replacing the password with a strong password as recommended by bitcoind. Do **not** use the password shown below. Create a file inside the .bitcoin directory, so that it is named .bitcoin/bitcoin.conf and enter a username and password:

```
rpcuser=bitcoinrpc
rpcpassword=7p687uGU8wMyBprB2aQrnt72r9Lh6jZy
```

Using bitcoind from the command line

The reference client bitcoind offers a number of commands that can be run from the command line. These are the same commands as those offered via the JSON-RPC API, so the command line allows us to experiment interactively with the capabilities that are also available programmatically. To start, we can invoke the help command to see a list of the available bitcoin commands:

Bitcoind command list.

```
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"
encryptwallet "passphrase"
getaccount "bitcoinaddress"
getaccountaddress "account"
getaddednodeinfo dns ( "node" )
getaddressesbyaccount "account"
getbalance ( "account" minconf )
getbestblockhash
getblock "hash" ( verbose )
getblockcount
getblockhash index
getblocktemplate ( "jsonrequestobject" )
getconnectioncount
getdifficulty
getgenerate
gethashespersec
getinfo
getmininginfo
getnettotals
getnetworkhashps ( blocks height )
getnewaddress ( "account" )
getpeerinfo
getrawchangeaddress
getrawmempool ( verbose )
getrawtransaction "txid" ( verbose )
getreceivedbyaccount "account" ( minconf )
getreceivedbyaddress "bitcoinaddress" ( minconf )
gettransaction "txid"
gettxout "txid" n ( includemempool )
gettxoutsetinfo
getunconfirmedbalance
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"
```

Running bitcoind

Commands: -daemon, getinfo

Now, run the bitcoin client. The first time you run it, it will rebuild the bitcoin blockchain. This is a multi-gigabyte file and will take on average 2 days to download in full. You can shorten the blockchain initialization time by downloading a partial copy of the blockchain using bittorrent from http://sourceforge.net/projects/bitcoin/files/Bitcoin/blockchain/.

Run bitcoind in the background with the option -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

[... more startup messages ...]
```

Bitcoin’s getinfo command shows us basic information about the status of the bitcoin network node, the wallet and the blockchain database:

```
$ bitcoind 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" : ""
}
```

The data is returned as a JavaScript Object Notation (JSON), a format which can easily be "consumed" by all programming languages but is also quite human-readable. Among this data we see the version of the bitcoin software client (9000), protocol (70002) and wallet file (60000). We see the current balance contained in the wallet, which is zero. We see the current block height, showing us how many blocks are known to this client, 286216. We also see various statistics about the bitcoin network and the settings related to this client. We will explore these settings in more detail in the rest of this chapter.

TIP

It will take some time, perhaps more than a day, for the bitcoind client to "catch up" to the current blockchain height as it downloads blocks from other bitcoin clients. You can check its current progress using getinfo to see the number of known blocks.

Wallet setup and encryption

Commands: bitcoind encryptwallet, walletpassphrase

Before we proceed with creating keys and other commands, we will first encrypt the wallet with a password. For this example, we use the encryptwallet command with the password "foo". Obviously, replace "foo" with a strong and complex password!

```
$ bitcoind 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.
$
```

We can verify the wallet has been encrypted, by running getinfo again. This time you will notice a new entry unlocked_until which is a counter showing how long the wallet decryption password will be stored in memory, keeping the wallet unlocked. At first this will be set to zero, meaning the wallet is locked:

```
$ bitcoind getinfo
{
    "version" : 90000,

[... other information...]

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

To unlock the wallet, we issue the walletpassphrase command that takes two parameters, the password and a number of seconds until the wallet is locked again automatically (a time counter):

```
$ bitcoind walletpassphrase foo 360
$
```

Confirm the wallet is unlocked and see the timeout by running getinfo again:

```
$ bitcoind getinfo
{
    "version" : 90000,

[... other information ...]

    "unlocked_until" : 1392580909,
    "errors" : ""
}
```

Wallet backup, plain-text dump and restore

Commands: backupwallet, importwallet, dumpwallet

Next, we will practice creating a wallet backup file and then restoring the wallet from the backup file. Use the backupwallet command to backup, providing the file name as the parameter. Here we backup the wallet to the file wallet.backup:

```
$ bitcoind backupwallet wallet.backup
$
```

Now, to restore the backup file, use the importwallet command. If your wallet is locked, you will need to unlock it first (see walletpassphrase above) in order to import the backup file:

```
$ bitcoind importwallet wallet.backup
$
```

The dumpwallet command can be used to dump the wallet into a text file that is human-readable:

```
$ bitcoind 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
[... many more keys ...]

$
```

Wallet addresses and receiving transactions

Commands: getnewaddress, getreceivedbyaddress, listtransactions, getaddressesbyaccount, getbalance

The bitcoin reference client maintains a pool of addresses, the size of which is displayed by keypoolsize when you use the command getinfo. These addresses are generated automatically and can then be used as public receiving addresses or change addresses. To get one of these addresses, you can use the getnewaddress command:

```
$ bitcoind getnewaddress
1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
```

Now, we can use this address to send a small amount of bitcoin to our bitcoind wallet from an external wallet (assuming you have some bitcoin in an exchange, web wallet or othe bitcoind wallet held elsewhere). For this example, we will send 50 millibits (0.050 bitcoin) to the address returned above.

We can now query the bitcoind client for the amount received by this address, and specify how many confirmations are required before an amount is counted in that balance. For this example, we will specify zero confirmations. A few seconds after sending the bitcoin from another wallet, we will see it reflected in the wallet. We use getreceivedbyaddress with the address and the number of confirmations set to zero (0):

```
$ bitcoind getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL 0
0.05000000
```

If we ommit the zero from the end of this command, we will only see the amounts that have at least minconf confirmations, where minconf is the setting for the minimum number of confirmations before a transaction is listed in the balance. The minconf setting is specified in the bitcoind configuration file. Since the transaction sending this bitcoin was only sent in the last few seconds, it has still not confirmed and therefore we will see it list a zero balance:

```
$ bitcoind getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
0.00000000
```

The transactions received by the entire wallet can also be displayed using the listtransactions command:

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

We can list all addresses in the entire wallet using the getaddressesbyaccount command:

```
$ bitcoind getaddressesbyaccount ""
[
    "1LQoTPYy1TyERbNV4zZbhEmgyfAipC6eqL",
    "17vrg8uwMQUibkvS2ECRX4zpcVJ78iFaZS",
    "1FvRHWhHBBZA8cGRRsGiAeqEzUmjJkJQWR",
    "1NVJK3JsL41BF1KyxrUyJW5XHjunjfp2jz",
    "14MZqqzCxjc99M5ipsQSRfieT7qPZcM7Df",
    "1BhrGvtKFjTAhGdPGbrEwP3xvFjkJBuFCa",
    "15nem8CX91XtQE8B1Hdv97jE8X44H3DQMT",
    "1Q3q6taTsUiv3mMemEuQQJ9sGLEGaSjo81",
    "1HoSiTg8sb16oE6SrmazQEwcGEv8obv9ns",
    "13fE8BGhBvnoy68yZKuWJ2hheYKovSDjqM",
    "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
    "1KHUmVfCJteJ21LmRXHSpPoe23rXKifAb2",
    "1LqJZz1D9yHxG4cLkdujnqG5jNNGmPeAMD"
]
```

Finally, the command getbalance will show the total balance of the wallet, adding up all transactions confirmed with at least minconf confirmations:

```
$ bitcoind getbalance
0.05000000
```

TIP

If the transaction has not yet confirmed, the balance returned by getbalance will be zero. The configuration option “minconf” determines the minimum number of confirmations that are required before a transaction shows in the balance

Exploring and decoding transactions

Commands: gettransaction, getrawtransaction, decoderawtransaction

We’ll now explore the incoming transaction that was listed above, using the gettransaction. We can retrieve a transaction by its transaction hash, shown at txid, above with the gettransaction command:

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

TIP

Transaction IDs are not authoritative until a transaction has been confirmed. Absence of a transaction hash in the blockchain does not mean the transaction was not processed. This is known as “transaction malleability”, as transaction hashes can be modified prior to confirmation in a block. After confirmation, the txid is immutable and authoritative.

The transaction form shown above with the command gettransaction is the simplified form. To retrieve the full transaction code and decode it we will use two commands, getrawtransaction and decoderawtransaction. First, getrawtransaction takes the transaction hash (txid) as a parameter and returns the full transaction as a “raw” hex string, exactly as it exists on the bitcoin network:

```
$ bitcoind getrawtransaction 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3
0100000001d717279515f88e2f56ce4e8a31e2ae3e9f00ba1d0add648e80c480ea22e0c7d3000000008b483045022100a4ebbeec83225dede
```

To decode this hex string, we can use the decoderawtransaction command. Copy and paste the hex as the first parameter of decoderawtransaction to get the full contents interpreted as a JSON data structure (for formatting reasons the hex string is shortened in the example below):

```
$ bitcoind decoderawtransaction 0100000001d717...388ac00000000
{
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "d3c7e022ea80c4808e64dd0a1dba009f3eaee2318a4ece562f8ef815952717d7",
            "vout" : 0,
            "scriptSig" : {
                "asm" :
"3045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a2df0dd0758e227383b302203301768ef878007e9ef7c304f70ffaf1f2c9
04793ac8a58ea751f9710e39aad2e296cc14daa44fa59248be58ede65e4c4b884ac5b5b6dede05ba84727e34c8fd3ee1d6929d7a44b6e111d

                "hex" :
"483045022100a4ebbeec83225dedead659bbde7da3d026c8b8e12e61a2df0dd0758e227383b302203301768ef878007e9ef7c304f70ffaf1f2

            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 0.05000000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY
OP_CHECKSIG",
                "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
                ]
            }
        },
        {
            "value" : 1.03362847,
            "n" : 1,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 107b7086b31518935c8d28703d66d09b36231343 OP_EQUALVERIFY
OP_CHECKSIG",
                "hex" : "76a914107b7086b31518935c8d28703d66d09b3623134388ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "12W9goQ3P7Waw5JH8fRVs1e2rVAKoGnvoy"
                ]
            }
        }
    ]
}
```

The transaction decode shows all the compoenents of this transaction, including the transaction inputs, and outputs. In this case we see that the transaction that credited our new address with 50 milibits used one input and generated two outputs. The input to this transaction was the output from a previously confirmed transaction (shown as the vin txid starting with d3c7 above). The two outputs correspond to the 50 milibit credit and an output with change back to the sender.

We can further explore the blockchain by examining the previous transaction referenced by its txid in this transaction, using the same commands (eg. gettransaction). Jumping from transaction to transaction we can follow a chain of transactions back as the coins are transmitted from owner address to owner address.

Once the transaction we received has been confirmed, by inclusion in a block, the gettransaction command will return additional information, showing the block hash (identifier) in which the transaction was included:

```
$ bitcoind gettransaction 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3
{
    "amount" : 0.05000000,
    "confirmations" : 1,
    "blockhash" : "000000000000000051d2e759c63a26e247f185ecb7926ed7a6624bc31c2a717b",
    "blockindex" : 18,
    "blocktime" : 1392660808,
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908,
    "details" : [
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.05000000
        }
    ]
}
```

Above, we see the new information in the entries blockhash, the hash of the block in which the transaction was included, and blockindex with value 18, indicating that our transaction was the 18th transaction in that block.

Exploring blocks

Commands: getblock, getblockhash

Now that we know which block our transaction was included in, we can query that block. We use the getblock command with the block hash as the parameter:

```
$ bitcoind getblock 000000000000000051d2e759c63a26e247f185ecb7926ed7a6624bc31c2a717b true
{
    "hash" : "000000000000000051d2e759c63a26e247f185ecb7926ed7a6624bc31c2a717b",
    "confirmations" : 2,
    "size" : 248758,
    "height" : 286384,
    "version" : 2,
    "merkleroot" : "9891747e37903016c3b77c7a0ef10acf467c530de52d84735bd55538719f9916",
    "tx" : [
        "46e130ab3c67d31d2b2c7f8fbc1ca71604a72e6bc504c8a35f777286c6d89bf0",
        "2d5625725b66d6c1da88b80b41e8c07dc5179ae2553361c96b14bcf1ce2c3868",
        "923392fc41904894f32d7c127059bed27dbb3cfd550d87b9a2dc03824f249c80",
        "f983739510a0f75837a82bfd9c96cd72090b15fa3928efb9cce95f6884203214",
        "190e1b010d5a53161aa0733b953eb29ef1074070658aaa656f933ded1a177952",
        "ee791ec8161440262f6e9144d5702f0057cef7e5767bc043879b7c2ff3ff5277",
        "4c45449ff56582664abfadeb1907756d9bc90601d32387d9cfd4f1ef813b46be",
        "3b031ed886c6d5220b3e3a28e3261727f3b4f0b29de5f93bc2de3e97938a8a53",
        "14b533283751e34a8065952fd1cd2c954e3d37aaa69d4b183ac6483481e5497d",
        "57b28365adaff61aaf60462e917a7cc9931904258127685c18f136eeaebd5d35",
        "8c0cc19fff6b66980f90af39bee20294bc745baf32cd83199aa83a1f0cd6ca51",
        "1b408640d54a1409d66ddaf3915a9dc2e8a6227439e8d91d2f74e704ba1cdae2",
        "0568f4fad1fdeff4dc70b106b0f0ec7827642c05fe5d2295b9deba4f5c5f5168",
        "9194bfe5756c7ec04743341a3605da285752685b9c7eebb594c6ed9ec9145f86",
        "765038fc1d444c5d5db9163ba1cc74bba2b4f87dd87985342813bd24021b6faf",
        "bff1caa9c20fa4eef33877765ee0a7d599fd1962417871ca63a2486476637136",
        "d76aa89083f56fcce4d5bf7fcf20c0406abdac0375a2d3c62007f64aa80bed74",
        "e57a4c70f91c8d9ba0ff0a55987ea578affb92daaa59c76820125f31a9584dfc",
        "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",

[... many more transactions ...]

    ],
    "time" : 1392660808,
    "nonce" : 3888130470,
    "bits" : "19015f53",
    "difficulty" : 3129573174.52228737,
    "chainwork" : "000000000000000000000000000000000000000000001931d1658fc04879e466",
    "previousblockhash" : "0000000000000000177e61d5f6ba6b9450e0dade9f39c257b4d48b4941ac77e7",
    "nextblockhash" : "0000000000000001239d2c3bf7f4c68a4ca673e434702a57da8fe0d829a92eb6"
}
```

The block contains 367 transactions and as you see above, the 18th transaction listed (9ca8f9…) is the txid of the one crediting 50 millibits to our address. The height entry tells us this is the 286384'th block in the blockchain.

We can also retrieve a block by its block height, using the getblockhash command, which takes the block height as the parameter and returns the block hash for that block:

```
$ bitcoind getblockhash 0
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Above, we retrieve the block hash of the "genesis block", the first block mined by Satoshi Nakamoto, at height zero. Retrieving this block shows:

```
$ bitcoind getblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
{
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations" : 286388,
    "size" : 285,
    "height" : 0,
    "version" : 1,
    "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx" : [
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "chainwork" : "0000000000000000000000000000000000000000000000000000000100010001",
    "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

The getblock, getblockhash and gettransaction commands can be used to explore the blockchain database, programmatically.

Creating, signing and submitting transactions based on unspent outputs

Commands: listunspent, gettxout, createrawtransaction, decoderawtransaction, signrawtransaction, sendrawtransaction

Bitcoin's transactions are based on the concept of spending "outputs", which are the result of previous transactions, creating a transaction chain that transfers ownership from address to address.

Our wallet has now received a transaction that assigned one such output to our address. Once this is confirmed, we can now spend that output.

First, we use the listunspent command to show all the unspent **confirmed** outputs in our wallet:

```
$ bitcoind listunspent
[
    {
        "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
        "vout" : 0,
        "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
        "account" : "",
        "scriptPubKey" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "amount" : 0.05000000,
        "confirmations" : 7
    }
]
```

We see that the transaction 9ca8f9… created an output (with vout index 0) assigned to the address 1hvzSo… for the amount of 50 millibits, which at this point has received 7 confirmations. Transactions use previously created outputs as their inputs, by referring to them by the previous txid and vout index. We will now create a transaction that will spend the 0'th vout of the txid 9ca8f9… as its input and assign it to a new output that sends value to a new address.

First, let's look at the specific output in more detail. We use the gettxout to get the details of this unspent output above. Transaction outputs are always referenced by txid and vout and these are the parameters we pass to gettxout:

```
$ bitcoind gettxout 9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3 0
{
    "bestblock" : "0000000000000001405ce69bd4ceebcdfdb537749cebe89d371eb37e13899fd9",
    "confirmations" : 7,
    "value" : 0.05000000,
    "scriptPubKey" : {
        "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY OP_CHECKSIG",
        "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
        "reqSigs" : 1,
        "type" : "pubkeyhash",
        "addresses" : [
            "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
        ]
    },
    "version" : 1,
    "coinbase" : false
}
```

What we see above is the output that assigned 50 millibits to our address 1hvz…. To spend this output we will create a new transaction. First, let's make an address to send the money to:

```
$ bitcoind getnewaddress
1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb
```

We will send 25 millibits to the new address 1LnfTn… we just created in our wallet. In our new transaction, we will spend the 50 millibit output and send 25 millibits to this new address. Because we have to spend the **whole** output from the previous transaction, we must also generate some change. We will generate change back to the 1hvz… address, sending the change back to the address from which the value originated. Finally, we will also have to pay a fee for this transaction. To pay the fee, we will reduce the change output by 0.5 millibits, and return 24.5 millibits in change. The difference between the sum of the new outputs (25mBTC + 24.5mBTC = 49.5mBTC) and the input (50mBTC) will be collected as a transaction fee by the miners.

We use the createrawtransaction to create the transaction described above. As parameters to createrawtransaction we provide the transaction input (the 50 millibit unspent output from our confirmed transaction) and the two transaction outputs (money sent to the new address and change sent back to the previous address):

```
bitcoind createrawtransaction
 '[{"txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3", "vout" : 0}]'
'{"1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb": 0.025,
"1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL": 0.0245}'

0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0000000000ffffffff02a02526000000000001976a9
```

The createrawtransaction command produces a raw hex string that encodes the transaction details we supplied. Let's confirm everything is correct by decoding this raw string using the decoderawtransaction command:

```
$ bitcoind decoderawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0000000000ffffffff02a02526000000000001976a9

{
    "txid" : "0793299cb26246a8d24e468ec285a9520a1c30fcb5b6125a102e3fc05d4f3cba",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
            "vout" : 0,
            "scriptSig" : {
                "asm" : "",
                "hex" : ""
            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 0.02500000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 d90d36e98f62968d2bc9bbd68107564a156a9bcf OP_EQUALVERIFY
OP_CHECKSIG",
                "hex" : "76a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb"
                ]
            }
        },
        {
            "value" : 0.02450000,
            "n" : 1,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY
OP_CHECKSIG",
                "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
                ]
            }
        }
    ]
}
```

That looks correct! Our new transaction “consumes” the unspent output from our confirmed transaction and then spends it in two outputs, one for 25 millibits to our new address and one for 24.5 millibits as change back to the original address. The difference of 0.5 millibits represents the transaction fee and will be credited to the miner who finds the block that includes our transaction.

As you may notice, the transaction contains an empty scriptSig, because we haven’t signed it yet. Without a signature, this transaction is meaningless, we haven’t yet proven that we **own** the address from which the unpsent output is sourced. By signing, we remove the encumberance on the output and prove that we own this output and can spend it. We use the signrawtransaction command to sign the transaction. It takes the raw transaction hex string as the parameter.

TIP

If the wallet is encrypted, you have to unlock it before you sign a transaction, as that operation requires access to the secret keys in your wallet

```
$ bitcoind walletpassphrase foo 360
$ bitcoind signrawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c0000000000ffffffff02a0252600000000001976a9

{
    "hex" :
"0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c000000006a47304402203e8a16522da80cef66b

    "complete" : true
}
```

The signrawtransaction command returns another hex encoded raw transaction. We decode it to see what changed, with decoderawtransaction:

```
$ bitcoind decoderawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c000000006a47304402203e8a16522da80cef66ba

{
    "txid" : "ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
            "vout" : 0,
            "scriptSig" : {
                "asm" :
"304402203e8a16522da80cef66bacfbc0c800c6d52c4a26d1d86a54e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e38
03c9700559f690c4a9182faa8bed88ad8a0c563777ac1d3f00fd44ea6c71dc5127",
                "hex" :
"47304402203e8a16522da80cef66bacfbc0c800c6d52c4a26d1d86a54e0a1b76d661f020c9022010397f00149f2a8fb2bc5bca52f2d7a7f87e

            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 0.02500000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 d90d36e98f62968d2bc9bbd68107564a156a9bcf OP_EQUALVERIFY
OP_CHECKSIG",
                "hex" : "76a914d90d36e98f62968d2bc9bbd68107564a156a9bcf88ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb"
                ]
            }
        },
        {
            "value" : 0.02450000,
            "n" : 1,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 07bdb518fa2e6089fd810235cf1100c9c13d1fd2 OP_EQUALVERIFY
OP_CHECKSIG",
                "hex" : "76a91407bdb518fa2e6089fd810235cf1100c9c13d1fd288ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL"
                ]
            }
        }
    ]
}
```

Now, the inputs used in the transaction contain a scriptSig, which is a digital signature proving ownership of address 1hvz… and removing the encumberance on the output so that it can be spent. The signature makes this transaction verifiable by any node in the bitcoin network.

Now it's time to submit the newly created transaction to the network. We do that with the command sendrawtransaction which takes the raw hex string produced by signrawtransaction, the same string we just decoded above:

```
$ bitcoind sendrawtransaction
0100000001e34ac1e2baac09c366fce1c2245536bda8f7db0f6685862aecf53ebd69f9a89c000000006a47304402203e8a16522da80cef66ba


ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346
```

The command sendrawtransaction returns a transaction hash (txid) as it submits the transaction on the network. We can now query that transaction id with gettransaction:

```
$ bitcoind gettransaction ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346
{
    "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
        }
    ]
}
```

As before, we can also examine this in more detail using the getrawtransaction and decodetransaction commands. These commands will return the exact same hex string that we produced and decoded previously just before we sent it on the network.

Alternative clients, libraries and toolkits

Beyond the reference client, bitcoind, there are other clients and libraries that can be used to interact with the bitcoin network and data structures. These are implemented in a variety of programming languages, offering programmers native interfaces in their own language.

Alternative implementations include:

- libbitcoin and sx tools, a C++ multi-threaded full node client and library with command-line tools (https://libbitcoin.dyne.org/)
- bitcoinj, a Java full node client library (https://code.google.com/p/bitcoinj/)
- btcd, a Go language full node bitcoin client (https://opensource.conformal.com/wiki/btcd)
- Bits of Proof (BOP), a Java enterprise-class implementation of bitcoin (https://bitsofproof.com)
- picocoin, a C implementation of a light-weight client library for bitcoin (https://github.com/jgarzik/picocoin)

Many more libraries exist in a variety of other programming languages and more are created all the time.

Libbitcoin and sx tools

The libbitcoin library is a C++ scalable multi-threaded and modular implemntation that supports a full-node client and a command-line toolset named "sx", which offers many of the same capabilities as the bitcoind client commands we illustrated in this chapter. The sx tools also offer some key management and manipulation tools that are not offered by bitcoind, including type-2 deterministic keys and key mnemonics.

Installing sx

To install sx and the supporting library libbitcoin, download and run the online installer on a Linux system:

```
$ wget http://sx.dyne.org/install-sx.sh
$ sudo bash ./install-sx.sh
```

You should now have the sx tools installed. Type sx with no parameters to display the help text, which lists all the available commands:

```
Usage: sx COMMAND [ARGS]...

  -c, --config              Specify a config file

The sx commands are:

DETERMINISTIC KEYS AND ADDRESSES
   genaddr                  Generate a Bitcoin address deterministically from a wallet
                         seed or master public key.
   genpriv                  Generate a private key deterministically from a seed.
   genpub                   Generate a public key deterministically from a wallet
                         seed or master public key.
   mpk                      Extract a master public key from a deterministic wallet seed.
   newseed                  Create a new deterministic wallet seed.

TRANSACTION PARSING
   showscript               Show the details of a raw script.
   showtx                   Show the details of a transaction.

BLOCKCHAIN QUERIES (blockexplorer.com)
   blke-fetch-transaction   Fetches a transaction from blockexplorer.com

FORMAT
   base58-decode            Convert from base58 to hex
   base58-encode            Convert from hex to base58
   base58check-decode       Convert from base58check to hex
   base58check-encode       Convert from hex to base58check
   decode-addr              Decode an address to its internal RIPEMD representation.
   embed-addr               Generate an address used for embedding record of data into the blockchain.
   encode-addr              Encode an address to base58check form.
   ripemd-hash              RIPEMD hash data from STDIN.
   unwrap                   Validates checksum and recovers version byte and original data from hexstring.
   validaddr                Validate an address.
   wrap                     Adds version byte and checksum to hexstring.

BRAINWALLET
   brainwallet              Make a private key from a brainwallet
   mnemonic                 Work with Electrum compatible mnemonics (12 words wallet seed).

BLOCKCHAIN WATCHING
   monitor                  Monitor an address.
   watchtx                  Watch transactions from the network searching for a certain hash.

BLOCKCHAIN QUERIES (blockchain.info)
   bci-fetch-last-height    Fetch the last block height using blockchain.info.
   bci-history              Get list of output points, values, and their spends
                         from blockchain.info

MISC
   btc                      Convert Satoshis into Bitcoins.
   initchain                Initialize a new blockchain.
   qrcode                   Generate Bitcoin QR codes offline.
   satoshi                  Convert Bitcoins into Satoshis.
   showblkhead              Show the details of a block header.
   wallet                   Experimental command line wallet.

MULTISIG ADDRESSES
   scripthash               Create BIP 16 script hash address from raw script hex.
```

```
LOOSE KEYS AND ADDRESSES
   addr                       See Bitcoin address of a public or private key.
   get-pubkey                 Get the pubkey of an address if available
   newkey                     Create a new private key.
   pubkey                     See the public part of a private key.

STEALTH
   secret-to-wif              Convert a secret exponent value to Wallet. Import. Format.
   stealth-new                Generate a new master stealth secret.
   stealth-recv               Regenerate the secret from your master secret and provided nonce.
   stealth-send               Generate a new sending address and a stealth nonce.

CREATE TRANSACTIONS
   mktx                       Create an unsigned tx.
   rawscript                  Create the raw hex representation from a script.
   set-input                  Set a transaction input.
   sign-input                 Sign a transaction input.

VALIDATE
   validsig                   Validate a transaction input's signature.

BLOCKCHAIN QUERIES
   balance                    Show balance of a Bitcoin address in satoshis.
   fetch-block-header         Fetch raw block header.
   fetch-last-height          Fetch the last block height.
   fetch-transaction          Fetch a raw transaction using a network connection to make requests against the obelisk load balancer
backend.
   fetch-transaction-index    Fetch block height and index in block of transaction.
   get-utxo                   Get enough unspent transaction outputs from a given set of
                              addresses to pay a given number of satoshis
   history                    Get list of output points, values, and their spends for an
                              address. grep can filter for just unspent outputs which can
                              be fed into mktx.
   validtx                    Validate a transaction.

BLOCKCHAIN UPDATES
   sendtx-bci                 Send tx to blockchain.info/pushtx.
   sendtx-node                Send transaction to a single node.
   sendtx-obelisk             Send tx to obelisk server.
   sendtx-p2p                 Send tx to bitcoin network.

See 'sx help COMMAND' for more information on a specific command.

SpesmiloXchange home page: <http://sx.dyne.org/>
```

Generating and manipulating keys with sxBitcoin Core

Generate a new private key, using the operating system’s random number generator, with the newkey command. We save the standard output into the file private_key:

```
$ sx newkey > private_key
$ cat private_key
5Jgx3UAaXw8AcCQCi1j7uaTaqpz2fqNR9K3r4apxdYn6rTzR1PL
```

Now, generate the public key from that private key, using the pubkey command. Pass the private_key file into the standard input and save the standard output of the command into a new file public_key:

```
$ sx pubkey < private_key > public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

We can re-format the public_key as an address, using the addr command. We pass the public_key into standard input:

```
$ sx addr < public_key
17re1S4Q8ZHyCP8Kw7xQad1Lr6XUzWUnkG
```

Deterministic keys with sx

The keys generated above are so called type-1 non-deterministic keys. That means that each one is generated from a random number generator. The sx tools also support type-2 deterministic keys, where a "master" key is created and then extended to produce a chain or tree of subkeys.

First, we generate a "seed" that will be used as the basis to derive a chain of keys, compatible with the Electrum wallet and other similar implementations. We use the newseed command to produce a seed value:

```
$ sx newseed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1
```

The seed value can also be exported as a word mnemonic that is human readable and easier to store and type than a hexadecimal string, using the mnemonic command:

```
$ sx mnemonic < seed > words
$ cat words
adore repeat vision worst especially veil inch woman cast recall dwell appreciate
```

The mnemonic words can be used to reproduce the seed, using the mnemonic command again:

```
$ sx mnemonic < words
eb68ee9f3df6bd4441a9feadec179ff1
```

With the seed, we can now generate a sequence of private and public keys, a key chain. We use the genpriv command to generate a sequence of private keys from a seed and the addr command to generate the corresponding public key.

```
$ sx genpriv 0 < seed
5JzY2cPZGViPGgXZ4Syb9Y4eUGjJpVt6sR8noxrpEcqgyj7LK7i
$ sx genpriv 0 < seed | sx addr
1esVQV2vR9JZPhFeRaeWkAhzmWq7Fi7t7
```

```
$ sx genpriv 1 < seed
5JdtL7ckAn3iFBFyVG1Bs3A5TqziFTaB9f8NeyNo8crnE2Sw5Mz
$ sx genpriv 1 < seed | sx addr
1G1oTeXitk76c2fvQWny4pryTdH1RTqSPW
```

With deterministic keys we can generate and re-generate thousands of keys, all derived from a single seed in a deterministic chain. This technique is used in many wallet applications to generate keys that can be backed up and restored with a simple multi-word mnemonic. This is easier than having to backup the wallet with all its randomly generated keys every time a new key is created.

TIP

The sx toolkit offers many useful commands for encoding and decoding addresses, converting to and from different formats and representations. Use them to explore the various formaat such as base58, base58check, hex etc.

About the Author

Andreas is a passionate technologist, who is well-versed in many technical subjects. He is a serial tech-entrepreneur, having launched businesses in London, New York, and California. He has earned degrees in Computer Science and Data Communications and Distributed Systems from UCL. With experience ranging from hardware and electronics to high level business and financial systems technology consulting and years as CTO/CIO/CSO in many companies — he combines authority and deep knowledge with an ability to make complex subjects easy to understand. More than 200 of his articles on security, cloud computing and data centers have been published in print and syndicated worldwide. His expertise includes Bitcoin, crypto-currencies, Information Security, Cryptography, Cloud Computing, Data Centers, Linux, Open Source and robotics software development. He also has been CISSP certified for 12 years.

As a bitcoin entrepreneur, Andreas has founded three bitcoin businesses and launched several community open-source projects. He often writes articles and blog posts on bitcoin, is a permanent host on Let's Talk Bitcoin and prolific public speaker at technology events. Andreas serves on the advisory boards of several bitcoin startups and serves as the Chief Security Officer of Blockchain.

Mastering Bitcoin

Andreas M. Antonopoulos

Editor
Allyson MacDonald

Editor
Mike Loukides

Revision History

2014-04-07
Early release revision 1

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472

2014-04-07T13:22:22-07:00