**2nd Edition**

Mastering

Bitcoin PROGRAMMING THE OPEN BLOCKCHAIN

Andreas M. Antonopoulos



**SECOND EDITION**

**Mastering Bitcoin**

***Programming the Open Blockchain Andreas M. Antonopoulos***

Beijing Boston Farnham Sebastopol Tokyo

**Mastering Bitcoin**

by Andreas M. Antonopoulos

Copyright © 2017 Andreas M. Antonopoulos, LLC. All rights reserved.

Printed in the United States of America.

Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O’Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/insti‐ tutional sales department: 800-998-9938 or *corporate@oreilly.com*.

**Editor:** Tim McGovern

**Production Editor:** Nicholas Adams **Copyeditor:** Kim Cofer

**Proofreader:** Christina Edwards June 2017: Second Edition

**Revision History for the Second Edition** 2017-06-01: First Release

2017-07-21: Second Release

**Indexer:** Judy McConville **Interior Designer:** David Futato **Cover Designer:** Randy Comer **Illustrator:** Rebecca Demarest

See *http://oreilly.com/catalog/errata.csp?isbn=9781491954386* for release details.

The O’Reilly logo is a registered trademark of O’Reilly Media, Inc. *Mastering Bitcoin*, the cover image, and related trade dress are trademarks of O’Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-95438-6

[LSI]

*Dedicated to my mum, eresa (1946–2017) She taught me to love books and question authority ank you, mum*

**Table of Contents**

**Preface. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xiii Quick Glossary. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxiii**

**1. Introduction. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1** What Is Bitcoin? 1 History of Bitcoin 4 Bitcoin Uses, Users, and Their Stories 5 Getting Started 6

Choosing a Bitcoin Wallet 6 Quick Start 8 Getting Your First Bitcoin 10 Finding the Current Price of Bitcoin 11 Sending and Receiving Bitcoin 12

**2. How Bitcoin Works. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15** Transactions, Blocks, Mining, and the Blockchain 15 Bitcoin Overview 15 Buying a Cup of Coffee 16 Bitcoin Transactions 18 Transaction Inputs and Outputs 18 Transaction Chains 19 Making Change 20 Common Transaction Forms 21 Constructing a Transaction 22 Getting the Right Inputs 22 Creating the Outputs 24 Adding the Transaction to the Ledger 25 Bitcoin Mining 26 Mining Transactions in Blocks 27

**v**

Spending the Transaction 29

**3. Bitcoin Core: The Reference Implementation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31** Bitcoin Development Environment 32 Compiling Bitcoin Core from the Source Code 32

Selecting a Bitcoin Core Release 33 Configuring the Bitcoin Core Build 34 Building the Bitcoin Core Executables 37

Running a Bitcoin Core Node 38 Running Bitcoin Core for the First Time 39 Configuring the Bitcoin Core Node 39

Bitcoin Core Application Programming Interface (API) 43 Getting Information on the Bitcoin Core Client Status 44 Exploring and Decoding Transactions 45 Exploring Blocks 47 Using Bitcoin Core’s Programmatic Interface 48

Alternative Clients, Libraries, and Toolkits 51 C/C++ 52 JavaScript 52 Java 52 Python 52 Ruby 53 Go 53 Rust 53 C# 53 Objective-C 53

**4. Keys, Addresses. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55** Introduction 55 Public Key Cryptography and Cryptocurrency 56 Private and Public Keys 57 Private Keys 58 Public Keys 60 Elliptic Curve Cryptography Explained 60 Generating a Public Key 63 Bitcoin Addresses 64 Base58 and Base58Check Encoding 66 Key Formats 70 Implementing Keys and Addresses in Python 76 Advanced Keys and Addresses 80 Encrypted Private Keys (BIP-38) 80 Pay-to-Script Hash (P2SH) and Multisig Addresses 81

**vi | Table of Contents**

Vanity Addresses 82 Paper Wallets 88

**5. Wallets. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93** Wallet Technology Overview 93 Nondeterministic (Random) Wallets 94 Deterministic (Seeded) Wallets 95 HD Wallets (BIP-32/BIP-44) 96 Seeds and Mnemonic Codes (BIP-39) 97 Wallet Best Practices 97 Using a Bitcoin Wallet 98 Wallet Technology Details 99 Mnemonic Code Words (BIP-39) 99 Creating an HD Wallet from the Seed 106 Using an Extended Public Key on a Web Store 110

**6. Transactions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117** Introduction 117 Transactions in Detail 117

Transactions—Behind the Scenes 118 Transaction Outputs and Inputs 119 Transaction Outputs 121 Transaction Inputs 123 Transaction Fees 126 Adding Fees to Transactions 129 Transaction Scripts and Script Language 131 Turing Incompleteness 131 Stateless Verification 132 Script Construction (Lock + Unlock) 132 Pay-to-Public-Key-Hash (P2PKH) 136 Digital Signatures (ECDSA) 138 How Digital Signatures Work 139 Verifying the Signature 141 Signature Hash Types (SIGHASH) 141 ECDSA Math 143 The Importance of Randomness in Signatures 145 Bitcoin Addresses, Balances, and Other Abstractions 145

**7. Advanced Transactions and Scripting. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 149** Introduction 149 Multisignature 149 Pay-to-Script-Hash (P2SH) 151

**Table of Contents | vii**

P2SH Addresses 153 Benefits of P2SH 154 Redeem Script and Validation 154

Data Recording Output (RETURN) 155 Timelocks 157 Transaction Locktime (nLocktime) 157 Check Lock Time Verify (CLTV) 158 Relative Timelocks 160 Relative Timelocks with nSequence 160 Relative Timelocks with CSV 162 Median-Time-Past 162 Timelock Defense Against Fee Sniping 163 Scripts with Flow Control (Conditional Clauses) 164 Conditional Clauses with VERIFY Opcodes 165 Using Flow Control in Scripts 166 Complex Script Example 167

**8. The Bitcoin Network. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171** Peer-to-Peer Network Architecture 171 Node Types and Roles 172 The Extended Bitcoin Network 173 Bitcoin Relay Networks 176 Network Discovery 176 Full Nodes 180 Exchanging “Inventory” 181 Simplified Payment Verification (SPV) Nodes 183 Bloom Filters 185

How Bloom Filters Work 186 How SPV Nodes Use Bloom Filters 189 SPV Nodes and Privacy 190 Encrypted and Authenticated Connections 191

Tor Transport 191 Peer-to-Peer Authentication and Encryption 191 Transaction Pools 192

**9. The Blockchain. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 195** Introduction 195 Structure of a Block 196 Block Header 197 Block Identifiers: Block Header Hash and Block Height 197 The Genesis Block 198 Linking Blocks in the Blockchain 200

**viii | Table of Contents**

Merkle Trees 201 Merkle Trees and Simplified Payment Verification (SPV) 207 Bitcoin’s Test Blockchains 207

Testnet—Bitcoin’s Testing Playground 208 Segnet—The Segregated Witness Testnet 210 Regtest—The Local Blockchain 210 Using Test Blockchains for Development 211

**10. Mining and Consensus. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 213** Introduction 213 Bitcoin Economics and Currency Creation 215 Decentralized Consensus 217 Independent Verification of Transactions 218 Mining Nodes 219 Aggregating Transactions into Blocks 220 The Coinbase Transaction 221 Coinbase Reward and Fees 223 Structure of the Coinbase Transaction 224 Coinbase Data 225 Constructing the Block Header 227 Mining the Block 228 Proof-of-Work Algorithm 228 Target Representation 235 Retargeting to Adjust Difficulty 235 Successfully Mining the Block 237 Validating a New Block 238 Assembling and Selecting Chains of Blocks 239 Blockchain Forks 240 Mining and the Hashing Race 247 The Extra Nonce Solution 249 Mining Pools 250 Consensus Attacks 253 Changing the Consensus Rules 256 Hard Forks 256 Hard Forks: Software, Network, Mining, and Chain 258 Diverging Miners and Difficulty 259 Contentious Hard Forks 260 Soft Forks 261 Criticisms of Soft Forks 262 Soft Fork Signaling with Block Version 262 BIP-34 Signaling and Activation 263 BIP-9 Signaling and Activation 264

**Table of Contents | ix**

Consensus Software Development 266

**11. Bitcoin Security. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 269** Security Principles 269 Developing Bitcoin Systems Securely 270 The Root of Trust 271 User Security Best Practices 272 Physical Bitcoin Storage 273 Hardware Wallets 273 Balancing Risk 273 Diversifying Risk 274 Multisig and Governance 274 Survivability 274 Conclusion 274

**12. Blockchain Applications. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 275** Introduction 275 Building Blocks (Primitives) 276 Applications from Building Blocks 278 Colored Coins 278

Using Colored Coins 279 Issuing Colored Coins 280 Colored Coins Transactions 280

Counterparty 283 Payment Channels and State Channels 284 State Channels—Basic Concepts and Terminology 285 Simple Payment Channel Example 286 Making Trustless Channels 289 Asymmetric Revocable Commitments 292 Hash Time Lock Contracts (HTLC) 296 Routed Payment Channels (Lightning Network) 297 Basic Lightning Network Example 298 Lightning Network Transport and Routing 301 Lightning Network Benefits 303 Conclusion 304

**A. The Bitcoin Whitepaper by Satoshi Nakamoto. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 305 x | Table of Contents**

**B. Transaction Script Language Operators, Constants, and Symbols. . . . . . . . . . . . . . . . . 317 C. Bitcoin Improvement Proposals. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 323 D. Segregated Witness. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 329 E. Bitcore. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 343 F. pycoin, ku, and tx. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 347 G. Bitcoin Explorer (bx) Commands. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 357 Index. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 361**

**Table of Contents | xi**

**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 six months, failing to grasp its importance. This is a reaction that I have seen repeated among many of the smartest people I know, which gives me some consolation. The second time I came across bit‐ coin, in a mailing list discussion, I decided to read the whitepaper 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 nine pages, when I realized that bit‐ coin was not simply a digital currency, but a network of trust that could also provide the basis for so much more than just currencies. The realization that “this isn’t money, it’s a decentralized trust network,” started me on a four-month journey to devour every scrap of information about bitcoin I could find. I became obsessed and enthral‐ led, spending 12 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 pounds lighter from lack of consistent meals, determined to dedicate myself to work‐ ing 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. Bit‐ coin was the topic that had driven me into a frenzy of creativity and consumed my thoughts; it was the most exciting technology I had encountered since the internet. It was now time to share my passion about this amazing technology with a broader audience.

**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

**xiii**

an in-depth introduction to bitcoin for noncoders—those trying to understand the inner workings of bitcoin and cryptocurrencies.

**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 fun‐ gus, which is the central food source for the colony. Get that? These ants are farming!

Although 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 sophisticated behavior exhibited by a multimillion-member colony is an emergent property from the interaction of the individuals in a social network.

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

Bitcoin is a highly sophisticated decentralized trust network that can support myriad financial processes. Yet, each node in the bitcoin network follows a few simple mathe‐ matical 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 follow‐ ing simple rules that together can do amazing things without any central coordina‐ tion.

**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 ele‐ ments 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. **xiv | Preface**

*Constant width italic*

Shows text that should be replaced with user-supplied values or by values deter‐ mined by context.

This icon signifies a tip or suggestion.

This icon signifies a general note.

This icon indicates a warning or caution.

**Code Examples**

The examples are illustrated in Python, C++, and using the command line of a Unix like operating system such as Linux or macOS. All code snippets are available in the Github repository (*https://github.com/bitcoinbook/bitcoinbook*) in the *code* subdirec‐ tory of the main repo. Fork the book code, try the code examples, or submit correc‐ tions via GitHub.

All the code snippets can be replicated on most operating systems with a minimal installation of compilers and interpreters for the corresponding languages. Where necessary, we provide basic installation instructions and step-by-step examples of the output of those instructions.

Some of the code snippets and code output have been reformatted for print. In all such cases, the lines have been split by a backslash (\) character, followed by a newline character. When transcribing the examples, remove those two characters and join the lines again and you should see identical results as shown in the example.

All the code snippets use real values and calculations where possible, so that you can build from example to example and see the same results in any code you write to cal‐ culate the same values. For example, the private keys and corresponding public keys and addresses are all real. The sample transactions, blocks, and blockchain references have all been introduced in the actual bitcoin blockchain and are part of the public ledger, so you can review them on any bitcoin system.

**Preface | xv**

**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 signifi‐ cant 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: “*Mastering Bitcoin* by Andreas M. Antonopoulos (O’Reilly). Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6.”

Some editions of this book are offered under an open source license, such as CC-BY NC, in which case the terms of that license apply.

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*.

**Bitcoin Addresses and Transactions in This Book**

The bitcoin addresses, transactions, keys, QR codes, and blockchain data used in this book are, for the most part, real. That means you can browse the blockchain, look at the transactions offered as examples, retrieve them with your own scripts or pro‐ grams, etc.

However, note that the private keys used to construct addresses are either printed in this book, or have been “burned.” That means that if you send money to any of these addresses, the money will either be lost forever, or in some cases everyone who can read the book can take it using the private keys printed in here.

DO NOT SEND MONEY TO ANY OF THE ADDRESSES IN 

THIS BOOK. Your money will be taken by another reader, or lost

forever.

**xvi | Preface**

**O’Reilly Safari**

*Safari* (formerly Safari Books Online) is a membership-based 

training and reference platform for enterprise, government,

educators, and individuals.

Members have access to thousands of books, training videos, Learning Paths, interac‐ tive tutorials, and curated playlists from over 250 publishers, including O’Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Profes‐ sional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, 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, and Course Technology, among others.

For more information, please visit *http://oreilly.com/safari*.

**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)

To comment or ask technical questions about this book, send email to *bookques‐ tions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our web‐ site 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*

**Preface | xvii**

**Contacting the Author**

You can contact me, Andreas M. Antonopoulos, on my personal site: *https://antono poulos.com/*

Information about *Mastering Bitcoin* as well as the Open Edition and translations are available on: *https://bitcoinbook.info/*

Follow me on Facebook: *https://facebook.com/AndreasMAntonopoulos* Follow me on Twitter: *https://twitter.com/aantonop*

Follow me on Linkedin: *https://linkedin.com/company/aantonop*

Many thanks to all my patrons who support my work through monthly donations. You can follow my Patreon page here: *https://patreon.com/aantonop*

**Acknowledgments**

This book represents the efforts and contributions of many people. I am grateful for all the help I received from friends, colleagues, and even complete strangers, who joined me in this effort to write the definitive technical book on cryptocurrencies and bitcoin.

It is impossible to make a distinction between the bitcoin technology and the bitcoin community, and this book is as much a product of that community as it is a book on the technology. My work on this book was encouraged, cheered on, supported, and rewarded by the entire bitcoin community from the very beginning until the very end. More than anything, this book has allowed me to be part of a wonderful com‐ munity for two years and I can’t thank you enough for accepting me into this com‐ munity. There are far too many people to mention by name—people I’ve met at conferences, events, seminars, meetups, pizza gatherings, and small private gather‐ ings, as well as many who communicated with me by Twitter, on reddit, on bitcoin‐ talk.org, and on GitHub who have had an impact on this book. Every idea, analogy, question, answer, and explanation you find in this book was at some point inspired, tested, or improved through my interactions with the community. Thank you all for your support; without you this book would not have happened. I am forever grateful.

The journey to becoming an author starts long before the first book, of course. My first language (and schooling) was Greek, so I had to take a remedial English writing course in my first year of university. I owe thanks to Diana Kordas, my English writ‐ ing teacher, who helped me build confidence and skills that year. Later, as a professio‐ nal, I developed my technical writing skills on the topic of data centers, writing for *Network World* magazine. I owe thanks to John Dix and John Gallant, who gave me my first writing job as a columnist at *Network World* and to my editor Michael Coo‐

**xviii | Preface**

ney and my colleague Johna Till Johnson who edited my columns and made them fit for publication. Writing 500 words a week for four years gave me enough experience to eventually consider becoming an author.

Thanks also to those who supported me when I submitted my book proposal to O’Reilly, by providing references and reviewing the proposal. Specifically, thanks to John Gallant, Gregory Ness, Richard Stiennon, Joel Snyder, Adam B. Levine, Sandra Gittlen, John Dix, Johna Till Johnson, Roger Ver, and Jon Matonis. Special thanks to Richard Kagan and Tymon Mattoszko, who reviewed early versions of the proposal and Matthew Taylor, who copyedited the proposal.

Thanks to Cricket Liu, author of the O’Reilly title *DNS and BIND*, who introduced me to O’Reilly. Thanks also to Michael Loukides and Allyson MacDonald at O’Reilly, who worked for months to help make this book happen. Allyson was especially patient when deadlines were missed and deliverables delayed as life intervened in our planned schedule. For the second edition, I thank Timothy McGovern for guiding the process, Kim Cofer for patiently editing, and Rebecca Panzer for illustrating many new diagrams.

The first few drafts of the first few chapters were the hardest, because bitcoin is a dif‐ ficult subject to unravel. Every time I pulled on one thread of the bitcoin technology, I had to pull on the whole thing. I repeatedly got stuck and a bit despondent as I struggled to make the topic easy to understand and create a narrative around such a dense technical subject. Eventually, I decided to tell the story of bitcoin through the stories of the people using bitcoin and the whole book became a lot easier to write. I owe thanks to my friend and mentor, Richard Kagan, who helped me unravel the story and get past the moments of writer’s block. I thank Pamela Morgan, who reviewed early drafts of each chapter in the first and second edition of the book, and asked the hard questions to make them better. Also, thanks to the developers of the San Francisco Bitcoin Developers Meetup group as well as Taariq Lewis and Denise Terry for helping test the early material. Thanks also to Andrew Naugler for info‐ graphic design.

During the development of the book, I made early drafts available on GitHub and invited public comments. More than a hundred comments, suggestions, corrections, and contributions were submitted in response. Those contributions are explicitly acknowledged, with my thanks, in “Early Release Draft (GitHub Contributions)” on page xx. Most of all, my sincere thanks to my volunteer GitHub editors Ming T. Nguyen (1st edition) and Will Binns (2nd edition), who worked tirelessly to curate, manage and resolve pull requests, issue reports, and perform bug fixes on GitHub.

Once the book was drafted, it went through several rounds of technical review. Thanks to Cricket Liu and Lorne Lantz for their thorough review, comments, and support.

**Preface | xix**

Several bitcoin developers contributed code samples, reviews, comments, and encouragement. Thanks to Amir Taaki and Eric Voskuil for example code snippets and many great comments; Chris Kleeschulte for contributing the Bitcore appendix; Vitalik Buterin and Richard Kiss for help with elliptic curve math and code contribu‐ tions; Gavin Andresen for corrections, comments, and encouragement; Michalis Kar‐ gakis for comments, contributions, and btcd writeup; and Robin Inge for errata submissions improving the second print. In the second edition, I again received a lot of help from many Bitcoin Core developers, including Eric Lombrozo who demysti‐ fied Segregated Witness, Luke Dashjr who helped improve the chapter on transac‐ tions, Johnson Lau who reviewed Segregated Witness and other chapters, and many others. I owe thanks to Joseph Poon, Tadge Dryja, and Olaoluwa Osuntokun who explained Lightning Network, reviewed my writing, and answered questions when I got stuck.

I owe my love of words and books to my mother, Theresa, who raised me in a house with books lining every wall. My mother also bought me my first computer in 1982, despite being a self-described technophobe. My father, Menelaos, a civil engineer who just published his first book at 80 years old, was the one who taught me logical and analytical thinking and a love of science and engineering.

Thank you all for supporting me throughout this journey.

**Early Release Draft (GitHub Contributions)**

Many contributors offered comments, corrections, and additions to the early-release draft on GitHub. Thank you all for your contributions to this book.

Following is a list of notable GitHub contributors, including their GitHub ID in parentheses:

• Alex Waters (alexwaters)

• Andrew Donald Kennedy (grkvlt)

• bitcoinctf

• Bryan Gmyrek (physicsdude)

• Casey Flynn (cflynn07)

• Chapman Shoop (belovachap)

• Christie D’Anna (avocadobreath)

• Cody Scott (Siecje)

• coinradar

• Cragin Godley (cgodley)

• dallyshalla

**xx | Preface**

• Diego Viola (diegoviola)

• Dirk Jäckel (biafra23)

• Dimitris Tsapakidis (dimitris-t)

• Dmitry Marakasov (AMDmi3)

• drstrangeM

• Ed Eykholt (edeykholt)

• Ed Leafe (EdLeafe)

• Edward Posnak (edposnak)

• Elias Rodrigues (elias19r)

• Eric Voskuil (evoskuil)

• Eric Winchell (winchell)

• Erik Wahlström (erikwam)

• effectsToCause (vericoin)

• Esteban Ordano (eordano)

• ethers

• fabienhinault

• Frank Höger (francyi)

• Gaurav Rana (bitcoinsSG)

• genjix

• halseth

• Holger Schinzel (schinzelh)

• Ioannis Cherouvim (cherouvim)

• Ish Ot Jr. (ishotjr)

• James Addison (jayaddison)

• Jameson Lopp (jlopp)

• Jason Bisterfeldt (jbisterfeldt)

• Javier Rojas (fjrojasgarcia)

• Jeremy Bokobza (bokobza)

• JerJohn15

• Joe Bauers (joebauers)

• joflynn

• Johnson Lau (jl2012)

**Preface | xxi**

• Jonathan Cross (jonathancross) • Jorgeminator

• Kai Bakker (kaibakker)

• Mai-Hsuan Chia (mhchia)

• Marzig (marzig76)

• Maximilian Reichel (phramz)

• Michalis Kargakis (kargakis)

• Michael C. Ippolito (michaelcippolito) • Mihail Russu (MihailRussu)

• Minh T. Nguyen (enderminh)

• Nagaraj Hubli (nagarajhubli)

• Nekomata (nekomata-3)

• Robert Furse (Rfurse)

• Richard Kiss (richardkiss)

• Ruben Alexander (hizzvizz)

• Sam Ritchie (sritchie)

• Sergej Kotliar (ziggamon)

• Seiichi Uchida (topecongiro)

• Simon de la Rouviere (simondlr) • Stephan Oeste (Emzy)

• takaya-imai

• Thiago Arrais (thiagoarrais)

• venzen

• Will Binns (wbnns)

• wintercooled

• wjx

• Wojciech Langiewicz (wlk)

• yurigeorgiev4

**xxii | Preface**

**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.

*address*

A bitcoin address looks like 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV. It consists of a string of letters and numbers. It’s really an encoded base58check version of a public key 160-bit hash. Just like you ask others to send an email to your email address, you would ask others to send you bitcoin to one of your bitcoin addresses.

*bip*

Bitcoin Improvement Proposals. A set of proposals that members of the bitcoin community have submitted to improve bitcoin. For example, BIP-21 is a pro‐ posal to improve the bitcoin uniform resource identifier (URI) scheme.

*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 produce a proof of work, thereby validating the transactions. Valid blocks are added to the main blockchain by net‐ work consensus.

*blockchain*

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

*Byzantine Generals Problem*

A reliable computer system must be able to cope with the failure of one or more of its components. A failed component may exhibit a type of behavior that is often overlooked—namely, sending conflicting information to different parts of

**xxiii**

the system. The problem of coping with this type of failure is expressed abstractly as the Byzantine Generals Problem.

*coinbase*

A special field used as the sole input for coinbase transactions. The coinbase allows claiming the block reward and provides up to 100 bytes for arbitrary data. Not to be confused with Coinbase transaction.

*coinbase transaction*

The first transaction in a block. Always created by a miner, it includes a single coinbase. Not to be confused with Coinbase.

*cold storage*

Refers to keeping a reserve of bitcoin offline. Cold storage is achieved when Bit‐ coin private keys are created and stored in a secure offline environment. Cold storage is important for anyone with bitcoin holdings. Online computers are vul‐ nerable to hackers and should not be used to store a significant amount of bit‐ coin.

*colored coins*

An open source Bitcoin 2.0 protocol that enables developers to create digital assets on top of bitcoin blockchain utilizing its functionalities beyond currency.

*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 confir‐ mations, and so on. Six or more confirmations is considered sufficient proof that a transaction cannot be reversed.

*consensus*

When several nodes, usually most nodes on the network, all have the same blocks in their locally-validated best block chain. Not to be confused with consensus rules.

*consensus rules*

The block validation rules that full nodes follow to stay in consensus with other nodes. Not to be confused with consensus.

*difficulty*

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

*difficulty retargeting*

A network-wide recalculation of the difficulty that occurs once every 2,016 blocks and considers the hashing power of the previous 2,016 blocks.

**xxiv | Quick Glossary**

*difficulty target*

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

*double spending*

Double spending is the result of successfully spending some money more than once. Bitcoin protects against double spending by verifying each transaction added to the block chain to ensure that the inputs for the transaction had not previously already been spent.

*ECDSA*

Elliptic Curve Digital Signature Algorithm or ECDSA is a cryptographic algo‐ rithm used by Bitcoin to ensure that funds can only be spent by their rightful owners.

*extra nonce*

As difficulty increased, miners often cycled through all 4 billion values of the nonce without finding a block. Because the coinbase script can store between 2 and 100 bytes of data, miners started using that space as extra nonce space, allow‐ ing them to explore a much larger range of block header values to find valid blocks.

*fees*

The sender of a transaction often includes a fee to the network for processing the requested transaction. Most transactions require a minimum fee of 0.5 mBTC.

*fork*

Fork, also known as accidental fork, occurs when two or more blocks have the same block height, forking the block chain. Typically occurs when two or more miners find blocks at nearly the same time. Can also happen as part of an attack.

*genesis block*

The first block in the blockchain, used to initialize the cryptocurrency.

*hard fork*

Hard fork, also known as Hard-Forking Change, is a permanent divergence in the blockchain, commonly occurs when non-upgraded nodes can’t validate blocks created by upgraded nodes that follow newer consensus rules. Not to be confused with fork, soft fork, software fork or Git fork.

*hardware wallet*

A hardware wallet is a special type of bitcoin wallet which stores the user’s private keys in a secure hardware device.

*hash*

A digital fingerprint of some binary input.

**Quick Glossary | xxv**

*hashlocks*

A hashlock is a type of encumbrance that restricts the spending of an output until a specified piece of data is publicly revealed. Hashlocks have the useful property that once any hashlock is opened publicly, any other hashlock secured using the same key can also be opened. This makes it possible to create multiple outputs that are all encumbered by the same hashlock and which all become spendable at the same time.

*HD protocol*

The Hierarchical Deterministic (HD) key creation and transfer protocol (BIP32), which allows creating child keys from parent keys in a hierarchy.

*HD wallet*

Wallets using the Hierarchical Deterministic (HD Protocol) key creation and transfer protocol (BIP32).

*HD wallet seed*

HD wallet seed or root seed is a potentially-short value used as a seed to generate the master private key and master chain code for an HD wallet.

*HTLC*

A Hashed TimeLock Contract or HTLC is a class of payments that use hashlocks and timelocks to require that the receiver of a payment either acknowledge receiving the payment prior to a deadline by generating cryptographic proof of payment or forfeit the ability to claim the payment, returning it to the payer.

*KYC*

Know your customer (KYC) is the process of a business, identifying and verify‐ ing the identity of its clients. The term is also used to refer to the bank regulation which governs these activities.

*LevelDB*

LevelDB is an open source on-disk key-value store. LevelDB is a light-weight, single-purpose library for persistence with bindings to many platforms.

*Lightning Networks*

Lightning Network is a proposed implementation of Hashed Timelock Contracts (HTLCs) with bi-directional payment channels which allows payments to be securely routed across multiple peer-to-peer payment channels. This allows the formation of a network where any peer on the network can pay any other peer even if they don’t directly have a channel open between each other.

*Locktime*

Locktime, or more technically nLockTime, is the part of a transaction which indicates the earliest time or earliest block when that transaction may be added to the block chain.

**xxvi | Quick Glossary**

*mempool*

The bitcoin Mempool (memory pool) is a collection of all transaction data in a block that have been verified by bitcoin nodes, but are not yet confirmed.

*merkle root*

The root node of a merkle tree, a descendant of all the hashed pairs in the tree. Block headers must include a valid merkle root descended from all transactions in that block.

*merkle tree*

A tree constructed by hashing paired data (the leaves), then pairing and hashing the results until a single hash remains, the merkle root. In Bitcoin, the leaves are almost always transactions from a single block.

*miner*

A network node that finds valid proof of work for new blocks, by repeated hash‐ ing.

*multisignature*

Multisignature (multisig) refers to requiring more than one key to authorize a bitcoin transaction.

*network*

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

*nonce*

The “nonce” in a bitcoin block is a 32-bit (4-byte) field whose value is set so that the hash of the block will contain a run of leading zeros. The rest of the fields may not be changed, as they have a defined meaning.

*off-chain transactions*

An off-chain transaction is the movement of value outside of the block chain. While an on-chain transaction—usually referred to as simply *a transaction*— modifies the blockchain and depends on the blockchain to determine its validity an off-chain transaction relies on other methods to record and validate the trans‐ action.

*opcode*

Operation codes from the Bitcoin Script language which push data or perform functions within a pubkey script or signature script.

*Open Assets protocol*

The Open Assets Protocol is a simple and powerful protocol built on top of the bitcoin blockchain. It allows issuance and transfer of user-created assets. The Open Assets protocol is an evolution of the concept of colored coins.

**Quick Glossary | xxvii**

*OP\_RETURN*

An opcode used in one of the outputs in an OP\_RETURN transaction. Not to be confused with OP\_RETURN transaction.

*OP\_RETURN transaction*

A transaction type relayed and mined by default in Bitcoin Core 0.9.0 and later that adds arbitrary data to a provably unspendable pubkey script that full nodes don’t have to store in their UTXO database. Not to be confused with OP\_RETURN opcode.

*orphan block*

Blocks whose parent block has not been processed by the local node, so they can’t be fully validated yet.

*orphan transactions*

Transactions that can’t go into the pool due to one or more missing input trans‐ actions.

*output*

Output, transaction output, or TxOut is an output in a transaction which con‐ tains two fields: a value field for transferring zero or more satoshis and a pubkey script for indicating what conditions must be fulfilled for those satoshis to be fur‐ ther spent.

*P2PKH*

Transactions that pay a bitcoin address contain P2PKH or Pay To PubKey Hash scripts. An output locked by a P2PKH script can be unlocked (spent) by present‐ ing a public key and a digital signature created by the corresponding private key.

*P2SH*

P2SH or Pay-to-Script-Hash is a powerful new type of transaction that greatly simplifies the use of complex transaction scripts. With P2SH the complex script that details the conditions for spending the output (redeem script) is not presen‐ ted in the locking script. Instead, only a hash of it is in the locking script.

*P2SH address*

P2SH addresses are Base58Check encodings of the 20-byte hash of a script, P2SH addresses use the version prefix “5”, which results in Base58Check-encoded addresses that start with a “3”. P2SH addresses hide all of the complexity, so that the person making a payment does not see the script.

*P2WPKH*

The signature of a P2WPKH (Pay-to-Witness-Public-Key-Hash) contains the same information as a P2PKH spending, but is located in the witness field instead of the scriptSig field. The scriptPubKey is also modified.

**xxviii | Quick Glossary**

*P2WSH*

The difference between P2SH and P2WSH (Pay-to-Witness-Script-Hash) is about the cryptographic proof location change from the scriptSig field to the wit‐ ness field and the scriptPubKey that is also modified.

*paper wallet*

In the most specific sense, a paper wallet is a document containing all of the data necessary to generate any number of Bitcoin private keys, forming a wallet of keys. However, people often use the term to mean any way of storing bitcoin off‐ line as a physical document. This second definition also includes paper keys and redeemable codes.

*payment channels*

A micropayment channel or payment channel is class of techniques designed to allow users to make multiple bitcoin transactions without committing all of the transactions to the bitcoin blockchain. In a typical payment channel, only two transactions are added to the block chain but an unlimited or nearly unlimited number of payments can be made between the participants.

*pooled mining*

Pooled mining is a mining approach where multiple generating clients contribute to the generation of a block, and then split the block reward according the con‐ tributed processing power.

*Proof-of-Stake*

Proof-of-Stake (PoS) is a method by which a cryptocurrency blockchain network aims to achieve distributed consensus. Proof-of-Stake asks users to prove owner‐ ship of a certain amount of currency (their “stake” in the currency).

*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 12.5 BTC per block.

*RIPEMD-160*

RIPEMD-160 is a 160-bit cryptographic hash function. RIPEMD-160 is a strengthened version of RIPEMD with a 160-bit hash result, and is expected to be secure for the next ten years or more.

**Quick Glossary | xxix**

*satoshi*

A satoshi is the smallest denomination of bitcoin that can be recorded on the blockchain. It is the equivalent of 0.00000001 bitcoin and is named after the crea‐ tor of Bitcoin, Satoshi Nakamoto.

*Satoshi Nakamoto*

Satoshi Nakamoto is the name used by the person or people who designed Bit‐ coin and created its original reference implementation, Bitcoin Core. As a part of the implementation, they also devised the first blockchain database. In the pro‐ cess they were the first to solve the double spending problem for digital currency. Their real identity remains unknown.

*Script*

Bitcoin uses a scripting system for transactions. Forth-like, Script is simple, stack-based, and processed from left to right. It is purposefully not Turing complete, with no loops.

*ScriptPubKey (aka pubkey script)*

ScriptPubKey or pubkey script, is a script included in outputs which sets the con‐ ditions that must be fulfilled for those satoshis to be spent. Data for fulfilling the conditions can be provided in a signature script.

*ScriptSig (aka signature script)*

ScriptSig or signature script, is the data generated by a spender which is almost always used as variables to satisfy a pubkey script.

*secret key (aka private key)*

The secret number that unlocks bitcoin sent to the corresponding address. A secret key looks like the following:

5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh

*Segregated Witness*

Segregated Witness is a proposed upgrade to the Bitcoin protocol which techno‐ logical innovation separates signature data from bitcoin transactions. Segregated Witness is a proposed soft fork; a change that technically makes Bitcoin’s proto‐ col rules more restrictive.

*SHA*

The Secure Hash Algorithm or SHA is a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST).

*simplified payment verification (SPV)*

SPV or simplified payment verification is a method for verifying particular trans‐ actions were included in a block without downloading the entire block. The method is used by some lightweight Bitcoin clients.

**xxx | Quick Glossary**

*soft fork*

soft fork or Soft-Forking Change is a temporary fork in the blockchain which commonly occurs when miners using non-upgraded nodes don’t follow a new consensus rule their nodes don’t know about. Not to be confused with fork, hard fork, software fork or Git fork.

*stale block*

Block which were successfully mined but which isn’t included on the current best block chain, likely because some other block at the same height had its chain extended first.

*timelocks*

A timelock is a type of encumbrance that restricts the spending of some bitcoin until a specified future time or block height. Timelocks feature prominently in many Bitcoin contracts, including payment channels and hashed timelock con‐ tracts.

*transaction*

In simple terms, a transfer of bitcoin from one address to another. More pre‐ cisely, 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.

*transaction pool*

An unordered collection of transactions that are not in blocks in the main chain, but for which we have input transactions.

*Turing completeness*

A program language is called “Turing complete,” if that it can run any program that a Turing machine can run given enough time and memory.

*unspent transaction output (UTXO)*

UTXO is an unspent transaction output that can be spent as an input in a new transaction.

*wallet*

Software that holds all your bitcoin addresses and secret keys. Use it to send, receive, and store your bitcoin.

*Wallet Import Format (WIF)*

WIF or Wallet Import Format is a data interchange format designed to allow exporting and importing a single private key with a flag indicating whether or not it uses a compressed public key.

Some contributed definitions have been sourced under a CC-BY license from the bit‐ coin Wiki or from other open source documentation sources.

**Quick Glossary | xxxi**

**CHAPTER 1**

**Introduction**

**What Is Bitcoin?**

Bitcoin is a collection of concepts and technologies that form the basis of a digital money ecosystem. Units of currency called bitcoin 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 trans‐ port 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, including buy and sell goods, send money to peo‐ ple or organizations, or extend credit. Bitcoin can be purchased, sold, and exchanged for other currencies at specialized currency exchanges. Bitcoin in a sense is the per‐ fect form of money for the internet because it is fast, secure, and borderless.

Unlike traditional currencies, bitcoin are entirely virtual. There are no physical coins or even digital coins per se. The coins are implied in transactions that transfer value from sender to recipient. Users of bitcoin own keys that allow them to prove owner‐ ship of bitcoin in the bitcoin network. With these keys they can sign transactions to unlock the value and spend it by transferring it to a new owner. Keys are often stored in a digital wallet on each user’s computer or smartphone. Possession of the key that can sign a transaction is the only prerequisite to spending bitcoin, putting the control entirely in the hands of each user.

Bitcoin is a distributed, peer-to-peer system. As such there is no “central” server or point of control. Bitcoin are created through a process called “mining,” which involves competing to find solutions to a mathematical problem while processing bit‐ coin transactions. Any participant in the bitcoin network (i.e., anyone using a device

**1**

running the full bitcoin protocol stack) may operate as a miner, using their comput‐ er’s processing power to verify and record transactions. Every 10 minutes, on average, a bitcoin miner is able to validate the transactions of the past 10 minutes and is rewarded with brand new bitcoin. Essentially, bitcoin mining decentralizes the currency-issuance and clearing functions of a central bank and replaces the need for any central bank.

The bitcoin protocol includes built-in algorithms that regulate the mining function across the network. The difficulty of the processing task that miners must perform is adjusted dynamically so that, on average, someone succeeds every 10 minutes regard‐ less of how many miners (and how much processing) are competing at any moment. The protocol also halves the rate at which new bitcoin are created every 4 years, and limits the total number of bitcoin that will be created to a fixed total just below 21 million coins. The result is that the number of bitcoin in circulation closely follows an easily predictable curve that approaches 21 million by the year 2140. Due to bitcoin’s diminishing rate of issuance, over the long term, the bitcoin currency is deflationary. Furthermore, bitcoin cannot be inflated by “printing” new money above and beyond the expected issuance rate.

Behind the scenes, bitcoin is also the name of the protocol, a peer-to-peer network, and a distributed computing innovation. The bitcoin currency is really only the first application of this invention. 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 decentralized peer-to-peer network (the bitcoin protocol)

• A public transaction ledger (the blockchain)

• A set of rules for independent transaction validation and currency issuance (con‐ sensus rules)

• A mechanism for reaching global decentralized consensus on the valid block‐ chain (Proof-of-Work algorithm)

As a developer, I see bitcoin as akin to the internet of money, a network for propagat‐ ing 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.

**2 | Chapter 1: Introduction**

| **Digital Currencies Before Bitcoin**  The emergence of viable digital money is closely linked to developments in cryptog‐ raphy. 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. Three basic questions for anyone accepting digital money are:  1. Can I trust that the money is authentic and not counterfeit?  2. Can I trust that the digital money can only be spent once (known as the “double spend” problem)?  3. Can I be sure that no one else can claim this money belongs to them and not me?  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 these cases, the counterfeiting and double-spend issues are handled by clearing all electronic transactions through central authorities that have a global view of the cur‐ rency 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 appro‐ priate architecture, digital signatures also can be used to address the double-spend issue.  When cryptography started becoming more broadly available and understood in the late 1980s, many researchers began trying to use cryptography to build digital curren‐ cies. These early digital currency projects issued digital money, usually backed by a national currency or precious metal such as gold.  Although these earlier digital currencies worked, they were centralized and, as a result, 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 tradi‐ tional 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, whether legitimate governments or crimi‐ nal elements, a *decentralized* digital currency was needed to avoid a single point of attack. Bitcoin is such a system, decentralized by design, and free of any central authority or point of control that can be attacked or corrupted. |
| --- |

**What Is Bitcoin? | 3**

**History of Bitcoin**

Bitcoin was invented in 2008 with the publication of a paper titled “Bitcoin: A Peer to-Peer Electronic Cash System,”1 written under the alias of Satoshi Nakamoto (see Appendix A). Nakamoto combined several prior inventions such as b-money and HashCash to create a completely decentralized electronic cash system that does not rely on a central authority for currency issuance or settlement and validation of trans‐ actions. The key innovation was to use a distributed computation system (called a “Proof-of-Work” algorithm) to conduct a global “election” every 10 minutes, allowing the decentralized network to arrive at *consensus* about the state of transactions. This elegantly solves the issue of double-spend where a single currency unit can be spent twice. Previously, the double-spend problem was a weakness of digital currency and was addressed 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. The implementation of the Proof-of-Work algorithm (mining) that provides security and resilience for bit‐ coin has increased in power exponentially, and now exceeds the combined processing power of the world’s top supercomputers. Bitcoin’s total market value has at times exceeded $35 billion US dollars, depending on the bitcoin-to-dollar exchange rate. The largest transaction processed so far by the network was $150 million US dollars, transmitted instantly and processed without any fees.

Satoshi Nakamoto withdrew from the public in April 2011, leaving the responsibility of developing the code and network to a thriving group of volunteers. The identity of the person or people behind bitcoin is still unknown. However, neither Satoshi Naka‐ moto nor anyone else exerts individual control over the bitcoin system, which oper‐ ates based on fully transparent mathematical principles, open source code, and consensus among participants. 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 and novel solution to a problem in distributed computing, known as the “Byzantine Generals’ Problem.” Briefly, the problem consists of trying to agree on a course of action or the state of a system 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 dis‐ tributed computing and has wide applicability beyond currency. It can be used to ach‐

1 “Bitcoin: A Peer-to-Peer Electronic Cash System,” Satoshi Nakamoto (*https://bitcoin.org/bitcoin.pdf*). **4 | Chapter 1: Introduction**

ieve consensus on decentralized networks to prove the fairness of elections, lotteries, asset registries, digital notarization, and more.

**Bitcoin Uses, Users, and Their Stories**

Bitcoin is an innovation in the ancient technology of money. At its core, money sim‐ ply facilitates the exchange of value between people. Therefore, in order to fully understand bitcoin and its uses, we’ll examine it from the perspective of people using it. Each of the people and their stories, as listed here, illustrates one or more specific use cases. We’ll be seeing them throughout the book:

*North American low-value retail*

Alice lives in Northern California’s 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 soft‐ ware, the exchanges, and basic transactions from the perspective of a retail con‐ sumer.

*North American high-value retail*

Carol is an art gallery owner in San Francisco. She sells expensive paintings for bitcoin. This story will introduce the risks of a “51%” consensus attack for retail‐ ers of high-value items.

*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 out‐ sourcing, contract services, and international wire transfers.

*Web store*

Gabriel is an enterprising young teenager in Rio de Janeiro, running a small web store that sells bitcoin-branded t-shirts, coffee mugs, and stickers. Gabriel is too young to have a bank account, but his parents are encouraging his entrepreneu‐ rial spirit.

*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.

**Bitcoin Uses, Users, and Their Stories | 5**

*Import/export*

Mohammed is an electronics importer in Dubai. He’s trying to use bitcoin to buy electronics from the United States and China for import into the UAE to acceler‐ ate 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 bitcoin 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.

Each of these stories is based on the real people and real industries currently using bitcoin to create new markets, new industries, and innovative solutions to global eco‐ nomic issues.

**Getting Started**

Bitcoin is a protocol that can be accessed using a client application that speaks the protocol. A “bitcoin wallet” is the most common user interface to the bitcoin system, just like a web browser is the most common user interface for the HTTP protocol. There are many implementations and brands of bitcoin wallets, just like there are many brands of web browsers (e.g., Chrome, Safari, Firefox, and Internet Explorer). And just like we all have our favorite browsers (Mozilla Firefox, Yay!) and our villains (Internet Explorer, Yuck!), bitcoin wallets vary in quality, performance, security, pri‐ vacy, and reliability. There is also a reference implementation of the bitcoin protocol that includes a wallet, known as the “Satoshi Client” or “Bitcoin Core,” which is derived from the original implementation written by Satoshi Nakamoto.

**Choosing a Bitcoin Wallet**

Bitcoin wallets are one of the most actively developed applications in the bitcoin eco‐ system. There is intense competition, and while a new wallet is probably being devel‐ oped right now, several wallets from last year are no longer actively maintained. Many wallets focus on specific platforms or specific uses and some are more suitable for beginners while others are filled with features for advanced users. Choosing a wallet is highly subjective and depends on the use and user expertise. It is therefore impossi‐ ble to recommend a specific brand or project of wallet. However, we can categorize bitcoin wallets according to their platform and function and provide some clarity about all the different types of wallets that exist. Better yet, moving money between bitcoin wallets is easy, cheap, and fast, so it is worth trying out several different wal‐ lets until you find one that fits your needs.

**6 | Chapter 1: Introduction**

Bitcoin wallets can be categorized as follows, according to the platform:

*Desktop wallet*

A desktop wallet was the first type of bitcoin wallet created as a reference imple‐ mentation and many users run desktop wallets for the features, autonomy, and control they offer. Running on general-use operating systems such as Windows and Mac OS has certain security disadvantages however, as these platforms are often insecure and poorly configured.

*Mobile wallet*

A mobile wallet is the most common type of bitcoin wallet. Running on smart phone operating systems such as Apple iOS and Android, these wallets are often a great choice for new users. Many are designed for simplicity and ease-of-use, but there are also fully featured mobile wallets for power users.

*Web wallet*

Web wallets 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. Some of these services operate using client-side code run‐ ning in the user’s browser, which keeps control of the bitcoin keys in the hands of the user. Most, however, present a compromise by taking control of the bitcoin keys from users in exchange for ease-of-use. It is inadvisable to store large amounts of bitcoin on third-party systems.

*Hardware wallet*

Hardware wallets are devices that operate a secure self-contained bitcoin wallet on special-purpose hardware. They are operated via USB with a desktop web browser or via near-field-communication (NFC) on a mobile device. By handling all bitcoin-related operations on the specialized hardware, these wallets are con‐ sidered very secure and suitable for storing large amounts of bitcoin.

*Paper wallet*

The keys controlling bitcoin can also be printed for long-term storage. These are known as paper wallets even though other materials (wood, metal, etc.) can be used. Paper wallets offer a low-tech but highly secure means of storing bitcoin long term. Offline storage is also often referred to as *cold storage*.

Another way to categorize bitcoin wallets is by their degree of autonomy and how they interact with the bitcoin network:

*Full-node client*

A full client, or “full node,” is a client that stores the entire history of bitcoin transactions (every transaction by every user, ever), manages users’ wallets, and can initiate transactions directly on the bitcoin network. A full node handles all aspects of the protocol and can independently validate the entire blockchain and

**Getting Started | 7**

any transaction. A full-node client consumes substantial computer resources (e.g., more than 125 GB of disk, 2 GB of RAM) but offers complete autonomy and independent transaction verification.

*Lightweight client*

A lightweight client, also known as a simple-payment-verification (SPV) client, connects to bitcoin full nodes (mentioned previously) for access to the bitcoin transaction information, but stores the user wallet locally and independently cre‐ ates, validates, and transmits transactions. Lightweight clients interact directly with the bitcoin network, without an intermediary.

*ird-party API client*

A third-party API client is one that interacts with bitcoin through a third-party system of application programming interfaces (APIs), rather than by connecting to the bitcoin network directly. The wallet may be stored by the user or by third party servers, but all transactions go through a third party.

Combining these categorizations, many bitcoin wallets fall into a few groups, with the three most common being desktop full client, mobile lightweight wallet, and web third-party wallet. The lines between different categories are often blurry, as many wallets run on multiple platforms and can interact with the network in different ways.

For the purposes of this book, we will be demonstrating the use of a variety of down‐ loadable bitcoin clients, from the reference implementation (Bitcoin Core) to mobile and web wallets. Some of the examples will require the use of Bitcoin Core, which, in addition to being a full client, also exposes APIs to the wallet, network, and transac‐ tion services. If you are planning to explore the programmatic interfaces into the bit‐ coin system, you will need to run Bitcoin Core, or one of the alternative clients (see “Alternative Clients, Libraries, and Toolkits” on page 51).

**Quick Start**

Alice, who we introduced in “Bitcoin Uses, Users, and Their Stories” on page 5, is not a technical user and only recently heard about bitcoin from her friend Joe. While at a party, Joe is once again enthusiastically explaining bitcoin to all around him and is offering a demonstration. Intrigued, Alice asks how she can get started with bitcoin. Joe says that a mobile wallet is best for new users and he recommends a few of his favorite wallets. Alice downloads “Mycelium” for Android and installs it on her phone.

When Alice runs Mycelium for the first time, as with many bitcoin wallets, the appli‐ cation automatically creates a new wallet for her. Alice sees the wallet on her screen, as shown in Figure 1-1 (note: do *not* send bitcoin to this sample address, it will be lost forever).

**8 | Chapter 1: Introduction**

|  |
| --- |

*Figure 1-1. e Mycelium Mobile Wallet*

The most important part of this screen is Alice’s *bitcoin address*. On the screen it appears as a long string of letters and numbers: 1Cdid9KFAaatwczBwBttQcw XYCpvK8h7FK. Next to the wallet’s bitcoin address is a QR code, a form of barcode that contains the same information in a format that can be scanned by a smartphone cam‐ era. The QR code is the square with a pattern of black and white dots. Alice can copy the bitcoin address or the QR code onto her clipboard by tapping the QR code, or the Receive button. In most wallets, tapping the QR code will also magnify it, so that it can be more easily scanned by a smartphone camera.

**Getting Started | 9**

Bitcoin addresses start with a 1 or 3. Like email addresses, they can 

be shared with other bitcoin users who can use them to send bit‐

coin directly to your wallet. There is nothing sensitive, from a secu‐

rity perspective, about the bitcoin address. It can be posted

anywhere without risking the security of the account. Unlike email

addresses, you can create new addresses as often as you like, all of

which will direct funds to your wallet. In fact, many modern wal‐

lets automatically create a new address for every transaction to

maximize privacy. A wallet is simply a collection of addresses and

the keys that unlock the funds within.

Alice is now ready to receive funds. Her wallet application randomly generated a pri‐ vate key (described in more detail in “Private Keys” on page 58) together with its cor‐ responding bitcoin address. 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. It was generated independently by her wallet without reference or registration with any service. In fact, in most wallets, there is no association between the bitcoin address and any externally identifiable information including the user’s identity. Until the moment this address is referenced as the recipient of value in a transaction posted on the bitcoin ledger, the bitcoin address is simply part of the vast number of possible addresses that are valid in bitcoin. Only once it has been associ‐ ated with a transaction does it become part of the known addresses in the network.

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

**Getting Your First Bitcoin**

The first and often most difficult task for new users is to acquire some bitcoin. Unlike other foreign currencies, you cannot yet buy bitcoin at a bank or foreign exchange kiosk.

Bitcoin transactions are irreversible. Most electronic payment networks such as credit cards, debit cards, PayPal, and bank account transfers are reversible. For someone selling bitcoin, this difference introduces a very high risk that the buyer will reverse the electronic payment after they have received bitcoin, in effect defrauding the seller. To mitigate this risk, companies accepting traditional electronic payments in return for bitcoin usually require buyers to undergo identity verification and credit worthiness checks, which may take several days or weeks. As a new user, this means you cannot buy bitcoin instantly with a credit card. With a bit of patience and crea‐ tive thinking, however, you won’t need to.

**10 | Chapter 1: Introduction**

Here are some methods for getting bitcoin as a new user:

• Find a friend who has bitcoin and buy some from him or her directly. Many bit‐ coin users start this way. This method is the least complicated. One way to meet people with bitcoin is to attend a local bitcoin meetup listed at Meetup.com.

• Use a classified service such as localbitcoins.com to find a seller in your area to buy bitcoin for cash in an in-person transaction.

• Earn bitcoin by selling a product or service for bitcoin. If you are a programmer, sell your programming skills. If you’re a hairdresser, cut hair for bitcoin.

• Use a bitcoin ATM in your city. A bitcoin ATM is a machine that accepts cash and sends bitcoin to your smartphone bitcoin wallet. Find a bitcoin ATM close to you using an online map from Coin ATM Radar.

• Use a bitcoin currency exchange linked to your bank account. Many countries now have currency exchanges that offer a market for buyers and sellers to swap bitcoin with local currency. Exchange-rate listing services, such as BitcoinAver‐ age, often show a list of bitcoin exchanges for each currency.

One of the advantages of bitcoin over other payment systems is 

that, when used correctly, it affords users much more privacy.

Acquiring, holding, and spending bitcoin does not require you to

divulge sensitive and personally identifiable information to third

parties. However, where bitcoin touches traditional systems, such

as currency exchanges, national and international regulations often

apply. In order to exchange bitcoin for your national currency, you

will often be required to provide proof of identity and banking

information. Users should be aware that once a bitcoin address is

attached to an identity, all associated bitcoin transactions are also

easy to identify and track. This is one reason many users choose to

maintain dedicated exchange accounts unlinked to their wallets.

Alice was introduced to bitcoin by a friend so she has an easy way to acquire her first bitcoin. Next, we will look at how she buys bitcoin from her friend Joe and how Joe sends the bitcoin to her wallet.

**Finding the Current Price of Bitcoin**

Before Alice can buy bitcoin from Joe, they have to agree on the *exchange rate* between bitcoin and US dollars. This brings up a common question for those new to bitcoin: “Who sets the bitcoin price?” The short answer is that the price is set by mar‐ kets.

**Getting Started | 11**

Bitcoin, like most other currencies, has a *floating exchange rate*. That means that the value of bitcoin vis-a-vis any other currency fluctuates according to supply and demand in the various markets where it is traded. For example, the “price” of bitcoin in US dollars is calculated in each market based on the most recent trade of bitcoin and US dollars. As such, the price tends to fluctuate minutely several times per sec‐ ond. A pricing service will aggregate the prices from several markets and calculate a volume-weighted average representing the broad market exchange rate of a currency pair (e.g., BTC/USD).

There are hundreds of applications and websites that can provide the current market rate. Here are some of the most popular:

*Bitcoin Average*

A site that provides a simple view of the volume-weighted-average for each cur‐ rency.

*CoinCap*

A service listing the market capitalization and exchange rates of hundreds of crypto-currencies, including bitcoin.

*Chicago Mercantile Exchange Bitcoin Reference Rate*

A reference rate that can be used for institutional and contractual reference, pro‐ vided as part of investment data feeds by the CME.

In addition to these various sites and applications, most bitcoin wallets will automati‐ cally convert amounts between bitcoin and other currencies. Joe will use his wallet to convert the price automatically before sending bitcoin to Alice.

**Sending and Receiving Bitcoin**

Alice has decided to exchange $10 US dollars for bitcoin, so as not to risk too much money on this new technology. She gives Joe $10 in cash, opens her Mycelium wallet application, and selects Receive. This displays a QR code with Alice’s first bitcoin address.

Joe then selects Send on his smartphone wallet and is presented with a screen con‐ taining two inputs:

• A destination bitcoin address

• The amount to send, in bitcoin (BTC) or his local currency (USD)

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, which is quite long and difficult to type. Joe taps the QR code icon and activates the smartphone camera, scanning the QR code displayed on Alice’s smartphone.

**12 | Chapter 1: Introduction**

Joe now has Alice’s bitcoin address set as the recipient. Joe enters the amount as $10 US dollars and his wallet converts it by accessing the most recent exchange rate from an online service. The exchange rate at the time is $100 US dollars per bitcoin, so $10 US dollars is worth 0.10 bitcoin (BTC), or 100 millibitcoin (mBTC) as shown in the screenshot from Joe’s wallet (see Figure 1-2).

|  |
| --- |

*Figure 1-2. Airbitz mobile bitcoin wallet send screen*

Joe then carefully checks to make sure he has entered the correct amount, because he is about to transmit money and mistakes are irreversible. After double-checking the address and amount, he presses Send to transmit the transaction. Joe’s mobile bitcoin wallet constructs a transaction that assigns 0.10 BTC to the address provided by Alice, sourcing the funds from Joe’s wallet and signing the transaction with Joe’s pri‐ vate keys. This tells the bitcoin network that Joe has authorized a transfer of value 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.

Meanwhile, Alice’s wallet is constantly “listening” to published transactions on the bitcoin network, looking for any that match the addresses in her wallets. A few sec‐

**Getting Started | 13**

onds after Joe’s wallet transmits the transaction, Alice’s wallet will indicate that it is receiving 0.10 BTC.

| **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 recorded in the bitcoin transaction ledger, known as the blockchain. To be confirmed, a transaction must be included in a block and added to the blockchain, which hap‐ pens every 10 minutes, on average. In traditional financial terms this is known as *clearing*. For more details on propagation, validation, and clearing (confirmation) of bitcoin transactions, see Chapter 10. |
| --- |

Alice is now the proud owner of 0.10 BTC that 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.

**14 | Chapter 1: Introduction**

**CHAPTER 2**

**How Bitcoin Works**

**Transactions, Blocks, Mining, and the Blockchain**

The bitcoin system, unlike traditional banking and payment systems, is based on decentralized 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. Subsequent chapters will delve into the technology behind transactions, the network, and mining.

**Bitcoin Overview**

In the overview diagram shown in Figure 2-1, we see that the bitcoin system consists of users with wallets containing keys, transactions that are propagated across the net‐ work, and miners who produce (through competitive computation) the consensus blockchain, which is the authoritative ledger of all transactions.

Each example in this chapter is based on an actual transaction made on the bitcoin network, simulating the interactions between the users (Joe, Alice, Bob, and Gopesh) by sending funds from one wallet to another. While tracking a transaction through the bitcoin network to the blockchain, we will use a *blockchain explorer* site to visual‐ ize 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.

**15**

|  |
| --- |

*Figure 2-1. Bitcoin overview*

Popular blockchain explorers include:

• Bitcoin Block Explorer

• BlockCypher Explorer

• blockchain.info

• BitPay Insight

Each of these has a search function that can take a bitcoin address, transaction hash, block number, or block hash and retrieve corresponding information from the bit‐ coin network. With each transaction or block example, we will provide a URL so you can look it up yourself and study it in detail.

**Buying a Cup of Coffee**

Alice, introduced in the previous chapter, is a new user who has just acquired her first bitcoin. In “Getting Your First Bitcoin” on page 10, Alice met with her friend 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 Cafe recently started accepting bitcoin payments by adding a bitcoin option to its point-of-sale system. 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 it into the register, as

**16 | Chapter 2: How Bitcoin Works**

he does for all transactions. The point-of-sale system automatically converts the total price from US dollars to bitcoin at the prevailing market rate and displays the price in both currencies:

Total:

$1.50 USD

0.015 BTC

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

Bob’s point-of-sale system will also automatically create a special QR code containing a *payment request* (see Figure 2-2).

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 prefill the information used to send the payment while showing a human-readable description to the user. You can scan the QR code with a bitcoin wallet application to see what Alice would see.

|  |
| --- |

*Figure 2-2. Payment request QR code*

Try to scan this with your wallet to see the address and amount but 

DO NOT SEND MONEY.

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 payment: "Purchase at Bob's Cafe"

**Transactions, Blocks, Mining, and the Blockchain | 17**

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 amount of time as a credit card authorization), Bob sees the transaction on the register, completing the transaction.

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

The bitcoin network can transact in fractional values, e.g., from 

millibitcoin (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 “bitcoin” to refer to any quantity of bitcoin currency,

from the smallest unit (1 satoshi) to the total number (21,000,000)

of all bitcoin that will ever be mined.

You can examine Alice’s transaction to Bob’s Cafe on the blockchain using a block explorer site (Example 2-1):

*Example 2-1. View Alice’s transaction on blockexplorer.com*

https://blockexplorer.com/tx/

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2 **Bitcoin Transactions**

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

**Transaction Inputs and Outputs**

Transactions are like lines in a double-entry bookkeeping ledger. Each transaction contains one or more “inputs,” which are like debits against a bitcoin account. On the other side of the transaction, there are one or more “outputs,” which are like credits added to a bitcoin account. The inputs and outputs (debits and credits) do not neces‐ sarily add up to the same amount. Instead, outputs add up to slightly less than inputs and the difference represents an implied *transaction fee*, which is a small payment col‐ lected by the miner who includes the transaction in the ledger. A bitcoin transaction is shown as a bookkeeping ledger entry in Figure 2-3.

The transaction also contains proof of ownership for each amount of bitcoin (inputs) whose value is being spent, in the form of a digital signature from the owner, which

**18 | Chapter 2: How Bitcoin Works**

can be independently validated by anyone. In bitcoin terms, “spending” is signing a transaction that transfers value from a previous transaction over to a new owner identified by a bitcoin address.

|  |
| --- |

*Figure 2-3. Transaction as double-entry bookkeeping*

**Transaction Chains**

Alice’s payment to Bob’s Cafe uses a previous transaction’s output as its input. In the previous chapter, Alice received bitcoin from her friend Joe in return for cash. That transaction created a bitcoin value locked by 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 that unlocks those previous transaction outputs, thereby proving to the bitcoin network that she owns the funds. She attaches the pay‐ ment for coffee to Bob’s address, thereby “encumbering” that output with the require‐ ment that Bob produces a signature in order to spend that amount. This represents a transfer of value between Alice and Bob. This chain of transactions, from Joe to Alice to Bob, is illustrated in Figure 2-4.

**Bitcoin Transactions | 19**

|  |
| --- |

*Figure 2-4. A chain of transactions, where the output of one transaction is the input of the next transaction*

**Making Change**

Many bitcoin transactions will include outputs that reference both an address of the new owner and an address of the current owner, called the *change* address. This is because transaction inputs, like currency notes, cannot be divided. If you purchase a $5 US dollar item in a store but use a $20 US dollar bill to pay for the item, you expect to receive $15 US dollars in change. The same concept applies with bitcoin transac‐ tion inputs. If you purchased an item that costs 5 bitcoin but only had a 20 bitcoin input to use, you would send one output of 5 bitcoin to the store owner and one out‐ put of 15 bitcoin back to yourself as change (less any applicable transaction fee). Importantly, the change address does not have to be the same address as that of the input and for privacy reasons is often a new address from the owner’s wallet.

Different wallets may use different strategies when aggregating inputs to make a pay‐ ment requested by the user. They might aggregate many small inputs, or use one that is equal to or larger than the desired payment. Unless the wallet can aggregate inputs in such a way to exactly match the desired payment plus transaction fees, the wallet will need to generate some change. This is very similar to how people handle cash. If you always use the largest bill in your pocket, you will end up with a pocket full of loose change. If you only use the loose change, you’ll always have only big bills. Peo‐ ple subconsciously find a balance between these two extremes, and bitcoin wallet developers strive to program this balance.

**20 | Chapter 2: How Bitcoin Works**

In summary, *transactions* move value from *transaction inputs* to *transaction outputs*. An input is a reference to a previous transaction’s output, showing where the value is coming from. A transaction output directs a specific value to a new owner’s bitcoin address and can include a change output back to the original owner. 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 owner to owner (see Figure 2-4).

**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 in Figure 2-5.

|  |
| --- |

*Figure 2-5. Most common transaction*

Another common form of transaction is one that aggregates several inputs into a sin‐ gle output (see Figure 2-6). 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 clean up lots of smaller amounts that were received as change for payments.

|  |
| --- |

*Figure 2-6. Transaction aggregating funds*

**Bitcoin Transactions | 21**

Finally, another transaction form that is seen often on the bitcoin ledger is a transac‐ tion that distributes one input to multiple outputs representing multiple recipients (see Figure 2-7). 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 it is completely offline. Like writing a check at home and later sending it to the bank in an envelope, the transaction does not need to be constructed and signed while con‐ nected to the bitcoin network.

**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 wallets keep track of all the available outputs belonging to addresses in the wallet. Therefore, Alice’s wallet would contain a copy of the transac‐ tion output from Joe’s transaction, which was created in exchange for cash (see “Get‐ ting Your First Bitcoin” on page 10). A bitcoin wallet application that runs as a full node 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 quickly verify incoming transactions as having correct inputs. However, because a full-node client takes up a lot of disk space, most user wallets run “lightweight” clients that track only the user’s own unspent outputs.

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-node using an application pro‐

**22 | Chapter 2: How Bitcoin Works**

gramming interface (API) call. Example 2-2 shows a API request, constructed as an HTTP GET command to a specific URL. This URL will return all the unspent trans‐ action outputs for an address, giving any application the information it needs to con‐ struct transaction inputs for spending. We use the simple command-line HTTP client *cURL* to retrieve the response.

*Example 2-2. Look up all the unspent outputs for Alice’s bitcoin address* $ 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 in Example 2-2 shows one unspent output (one that has not been redeemed yet) under the ownership of Alice’s address 1Cdid9KFAaatwczBwBttQcw XYCpvK8h7FK. The response includes the reference to the transaction in which this unspent output is contained (the payment from Joe) and its 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.

View the transaction from Joe to Alice.

As you can see, Alice’s wallet contains enough bitcoin 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 the 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).

**Constructing a Transaction | 23**

**Creating the Outputs**

A transaction output is created in the form of a script that creates an encumbrance 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 correspond‐ ing to Bob’s public address.” Because only Bob has the wallet with the keys corre‐ sponding 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 sig‐ nature from Bob.

This transaction will also include a second output, because Alice’s funds are in 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 as an output in the very same transaction as the payment to Bob. Essentially, Alice’s wal‐ let breaks her funds into two payments: one to Bob and one back to herself. She can then use (spend) the change output in a subsequent transaction.

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, because they will add up to less than 0.10. The resulting difference is the *transaction fee* that is collected by the miner as a fee for validating and including the transaction in a block to be recorded on the blockchain.

The resulting transaction can be seen using a blockchain explorer web application, as shown in Figure 2-8.

|  |
| --- |

*Figure 2-8. Alice’s transaction to Bob’s Cafe*

**24 | Chapter 2: How Bitcoin Works**

****View the transaction from Alice to Bob’s Cafe.

**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 owners. Now, the transaction must be transmitted to the bitcoin network where it will become part of 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**

Because 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**

Any system, such as a server, desktop application, or wallet, that participates in the bitcoin network by “speaking” the bitcoin protocol is called a *bitcoin node*. Alice’s wal‐ let application can send the new transaction to any bitcoin node it is connected to over any type of connection: wired, WiFi, mobile, etc. 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 node that receives a valid transaction it has not seen before will immedi‐ ately forward it to all other nodes to which it is connected, a propagation technique known as *flooding*. 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, Bob’s wallet application might be the first node to receive the transaction. However, even if Alice’s wallet sends the transaction through other nodes, it 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,

**Constructing a Transaction | 25**

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.

A common misconception about bitcoin transactions is that they 

must be “confirmed” by waiting 10 minutes for a new block, or up

to 60 minutes for a full six confirmations. Although confirmations

ensure the transaction has been accepted by the whole network,

such a delay is unnecessary for small-value items such as a cup of

coffee. A merchant may accept a valid small-value transaction with

no confirmations, with no more risk than a credit card payment

made without an ID or a signature, as merchants routinely accept

today.

**Bitcoin Mining**

Alice’s transaction is now propagated on the bitcoin network. It does not become part of the *blockchain* until it is verified and included in a block by a process called *mining*. See Chapter 10 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. The mining process serves two purposes in bitcoin:

• Mining nodes validate all transactions by reference to bitcoin’s *consensus rules*. Therefore, mining provides security for bitcoin transactions by rejecting invalid or malformed transactions.

• Mining creates new bitcoin in each block, almost like a central bank printing new money. The amount of bitcoin created per block is limited and diminishes with time, following a fixed issuance schedule.

Mining achieves a fine balance between cost and reward. Mining uses electricity to solve a mathematical problem. A successful miner will collect a *reward* in the form of new bitcoin and transaction fees. However, the reward will only be collected if the miner has correctly validated all the transactions, to the satisfaction of the rules of *consensus*. This delicate balance provides security for bitcoin without a central authority.

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 puz‐ zle, several thousand rows and columns in size. If I show you a completed puzzle you can verify it quite quickly. However, if the puzzle has a few squares filled and the rest

**26 | Chapter 2: How Bitcoin Works**

are empty, 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 crypto‐ graphic hash and exhibits similar characteristics: it is asymmetrically hard to solve but easy to verify, and its difficulty can be adjusted.

In “Bitcoin Uses, Users, and Their Stories” on page 5, we introduced Jing, an entre‐ preneur in Shanghai. Jing runs a *mining farm*, which is a business that runs thousands of specialized mining computers, competing for the reward. Every 10 minutes or so, Jing’s mining computers compete against thousands of similar systems in a global race to find a solution to a block of transactions. Finding such a solution, the so called *Proof-of-Work* (PoW), requires quadrillions of hashing operations per second across the entire bitcoin network. The algorithm for Proof-of-Work involves repeat‐ edly hashing the header of the block and a random number with the SHA256 crypto‐ graphic algorithm until a solution matching a predetermined 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 high-end dedicated graphical processing units (GPUs) cards such as those used in gaming desktops or consoles. At the time of this writing, the difficulty is so high that it is profitable only to mine with application specific integrated circuits (ASIC), essentially hundreds of mining algorithms printed in hardware, running in parallel on a single silicon chip. Jing’s company also partici‐ pates in a *mining pool*, which much like a lottery pool allows several participants to share their efforts and rewards. Jing’s company now runs a warehouse containing thousands of ASIC miners to mine for bitcoin 24 hours a day. The company pays its electricity costs by selling the bitcoin it is able to generate from mining, creating some income from the profits.

**Mining Transactions in Blocks**

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 tem‐ porary pool of unverified transactions maintained by each node. As miners construct a new block, they add unverified transactions from this pool to the new block and then attempt to prove the validity of that new block, with the mining algorithm (Proof-of-Work). The process of mining is explained in detail in Chapter 10.

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 he receives the previous block from the network, knowing he

**Mining Transactions in Blocks | 27**

has lost that previous round of competition. He immediately creates a new block, fills it with transactions and the fingerprint of the previous block, and starts calculating the Proof-of-Work for the new block. Each miner includes a special transaction in his block, one that pays his own bitcoin address the block reward (currently 12.5 newly created bitcoin) plus the sum of transaction fees from all the transactions included in the block. If he finds a solution that makes that block valid, he “wins” this reward because his successful block is added to the global blockchain and the reward transac‐ tion he included becomes spendable. Jing, who participates in a mining pool, has set up 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 unveri‐ fied transactions. Once validated by the mining software it was included in a new block, called a *candidate block*, generated by Jing’s mining pool. All the miners partici‐ pating in that mining pool immediately start computing Proof-of-Work for the can‐ didate block. Approximately five minutes after the transaction was first transmitted by Alice’s wallet, one of Jing’s ASIC miners found a solution for the candidate block and announced it to the network. Once other miners validated the winning block they started the race to generate the next block.

Jing’s winning block became part of the blockchain as block #277316, containing 420 transactions, including Alice’s transaction. The block containing Alice’s transaction is counted as one “confirmation” of that transaction.

You can see the block that includes Alice’s transaction.

Approximately 19 minutes later, a new block, #277317, is mined by another miner. Because this new block is built on top of block #277316 that contained Alice’s transac‐ tion, it added even more computation to the blockchain, thereby strengthening the trust in those transactions. Each block mined on top of the one containing the trans‐ action counts as an additional confirmation for Alice’s 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 in Figure 2-9, we can see block #277316, which contains Alice’s trans‐ action. Below it are 277,316 blocks (including block #0), linked to each other in a chain of blocks (blockchain) all the way back to block #0, known as 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

**28 | Chapter 2: How Bitcoin Works**

Alice’s transaction act as further assurance, as they pile on more computation in a longer and longer chain. By convention, any block with more than six confirmations is considered irrevocable, because it would require an immense amount of computa‐ tion to invalidate and recalculate six blocks. We will examine the process of mining and the way it builds trust in more detail in Chapter 10.

|  |
| --- |

*Figure 2-9. Alice’s transaction included in block #277316*

**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-node clients can track the source of the funds from the moment the bitcoin were first generated in a block, incrementally from transaction to transaction, until they reach Bob’s address. Lightweight clients can do what is called a simplified payment verification (see “Simplified Payment Verification (SPV) Nodes” on page 183) by con‐ firming that the transaction is in the blockchain and has several blocks mined after it, thus providing assurance that the miners accepted it as valid.

Bob can now spend the output from this and other transactions. 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 aggregate the various payments into a single

**Spending the Transaction | 29**

output (and a single address). For a diagram of an aggregating transaction, see Figure 2-6.

As Bob spends the payments received from Alice and other customers, he extends the chain of transactions. Let’s assume that Bob pays his web designer Gopesh in Banga‐ lore for a new website page. Now the chain of transactions will look like Figure 2-10.

|  |
| --- |

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

In this chapter, we saw how transactions build a chain that moves value from owner to owner. We also tracked Alice’s transaction, from the moment it was created in her wallet, through the bitcoin network and to the miners who recorded it on the block‐ chain. In the rest of this book we will examine the specific technologies behind wal‐ lets, addresses, signatures, transactions, the network, and finally mining.

**30 | Chapter 2: How Bitcoin Works**

**CHAPTER 3**

**Bitcoin Core: The Reference**

**Implementation**

Bitcoin is an *open source* project and the source code is available under an open (MIT) license, free to download and use for any purpose. Open source means more than simply free to use. It also means that bitcoin is developed by an open commu‐ nity of volunteers. At first, that community consisted of only Satoshi Nakamoto. By 2016, bitcoin’s source code had more than 400 contributors with about a dozen devel‐ opers working on the code almost full-time and several dozen more on a part-time basis. Anyone can contribute to the code—including you!

When bitcoin was created by Satoshi Nakamoto, the software was actually completed before the whitepaper reproduced in Appendix A was written. Satoshi wanted to make sure it worked before writing about it. That first implementation, then simply known as “Bitcoin” or “Satoshi client,” has been heavily modified and improved. It has evolved into what is known as *Bitcoin Core*, to differentiate it from other compatible implementations. Bitcoin Core is the *reference implementation* of the bitcoin system, meaning that it is the authoritative reference on how each part of the technology should be implemented. Bitcoin Core implements all aspects of bitcoin, including wallets, a transaction and block validation engine, and a full network node in the peer-to-peer bitcoin network.

Even though Bitcoin Core includes a reference implementation of a 

wallet, this is not intended to be used as a production wallet for

users or for applications. Application developers are advised to

build wallets using modern standards such as BIP-39 and BIP-32

(see “Mnemonic Code Words (BIP-39)” on page 99 and “HD Wal‐

lets (BIP-32/BIP-44)” on page 96). BIP stands for *Bitcoin Improve‐*

*ment Proposal*.

**31**

Figure 3-1 shows the architecture of Bitcoin Core.

|  |
| --- |

*Figure 3-1. Bitcoin Core architecture (Source: Eric Lombrozo)*

**Bitcoin Development Environment**

If you’re a developer, you will want to set up a development environment with all the tools, libraries, and support software for writing bitcoin applications. In this highly technical chapter, we’ll walk through that process step-by-step. If the material becomes too dense (and you’re not actually setting up a development environment) feel free to skip to the next chapter, which is less technical.

**Compiling Bitcoin Core from the Source Code**

Bitcoin Core’s source code can be downloaded as a ZIP archive or by cloning the authoritative source repository from GitHub. On the GitHub bitcoin page, select

**32 | Chapter 3: Bitcoin Core: The Reference Implementation**

Download ZIP from the sidebar. Alternatively, use the git command line to create a local copy of the source code on your system.

In many of the examples in this chapter we will be using the oper‐ 

ating system’s command-line interface (also known as a “shell”),

accessed via a “terminal” application. The shell will display a

prompt; you type a command; and the shell responds with some

text and a new prompt for your next command. The prompt may

look different on your system, but in the following examples it is

denoted by a $ symbol. In the examples, when you see text after a $

symbol, don’t type the $ symbol but type the command immedi‐

ately following it, then press Enter to execute the command. In the

examples, the lines below each command are the operating system’s

responses to that command. When you see the next $ prefix, you’ll

know it’s a new command and you should repeat the process.

In this example, we are using the git command to create a local copy (“clone”) of the source code:

$ git clone https://github.com/bitcoin/bitcoin.git

Cloning into 'bitcoin'...

remote: Counting objects: 66193, done.

remote: Total 66193 (delta 0), reused 0 (delta 0), pack-reused 66193 Receiving objects: 100% (66193/66193), 63.39 MiB | 574.00 KiB/s, done. Resolving deltas: 100% (48395/48395), done.

Checking connectivity... done.

$

Git is the most widely used distributed version control system, an 

essential part of any software developer’s toolkit. You may need to

install the git command, or a graphical user interface for git, on

your operating system if you do not have it already.

When the git cloning operation has completed, 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

**Selecting a Bitcoin Core Release**

By default, the local copy will be synchronized with the most recent code, which might be an unstable or beta version of bitcoin. Before compiling the code, 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

**Compiling Bitcoin Core from the Source Code | 33**

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

v0.10.0

...

v0.11.2

v0.11.2rc1

v0.12.0rc1

v0.12.0rc2

...

The list of tags shows all the released versions of bitcoin. By convention, *release candi‐ dates*, which are intended for testing, have the suffix “rc.” Stable releases that can be run on production systems have no suffix. From the preceding list, select the highest version release, which at the time of writing was v0.11.2. To synchronize the local code with this version, use the git checkout command:

$ git checkout v0.11.2

HEAD is now at 7e27892... Merge pull request #6975

You can confirm you have the desired version “checked out” by issuing the command git status:

$ git status

HEAD detached at v0.11.2

nothing to commit, working directory clean

**Configuring the Bitcoin Core Build**

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 and using the spacebar 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 instruc‐ tions for macOS and Windows can be found in the *doc* directory, as *build-osx.md* or *build-windows.md*, respectively.

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

**34 | Chapter 3: Bitcoin Core: The Reference Implementation**

The Bitcoin Core 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 follow‐

ing example. 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 following examples.

$ ./autogen.sh

...

glibtoolize: copying file 'build-aux/m4/libtool.m4'

glibtoolize: copying file 'build-aux/m4/ltoptions.m4'

glibtoolize: copying file 'build-aux/m4/ltsugar.m4'

glibtoolize: copying file 'build-aux/m4/ltversion.m4'

...

configure.ac:10: installing 'build-aux/compile'

configure.ac:5: installing 'build-aux/config.guess'

configure.ac:5: installing 'build-aux/config.sub'

configure.ac:9: installing 'build-aux/install-sh'

configure.ac:9: installing 'build-aux/missing'

Makefile.am: installing 'build-aux/depcomp'

...

The *autogen.sh* script creates a set of automatic configuration scripts that will inter‐ rogate your system to discover the correct settings and ensure you have all the neces‐ sary 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 0.11.2 to adapt to many kinds of systems. Usage: ./configure [OPTION]... [VAR=VALUE]...

...

Optional Features:

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

--enable-wallet enable wallet (default is yes)

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

...

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. In this chapter, we will build the bitcoind client with all the default features. We won’t be using the

**Compiling Bitcoin Core from the Source Code | 35**

configuration flags, but you should review them to understand what optional features are part of the client. If you are in an academic setting, computer lab restrictions may require you to install applications in your home directory (e.g., using --prefix= $HOME).

Here are some useful options that override the default behavior of the configure script:

--prefix=$HOME

This overrides the default installation location (which is */usr/local/*) for the resulting executable. Use $HOME to put everything in your home directory, or a different path.

--disable-wallet

This is used to disable the reference wallet implementation.

--with-incompatible-bdb

If you are building a wallet, allow the use of an incompatible version of the Berkeley DB library.

--with-gui=no

Don’t build the graphical user interface, which requires the Qt library. This builds server and command-line bitcoin only.

Next, run the configure script to automatically discover all the necessary libraries and create a customized build script for your 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... gawk

checking whether make sets $(MAKE)... yes

...

[many pages of configuration tests follow]

...

$

If all went 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. If an error occurs, it is most likely because of a missing or incompatible library. Review the build documentation again and make sure you install the missing prerequisites. Then run configure again and see if that fixes the error.

**36 | Chapter 3: Bitcoin Core: The Reference Implementation**

**Building the Bitcoin Core Executables**

Next, you will compile the source code, a process that can take up to an hour to com‐ plete, depending on the speed of your CPU and available memory. During the compi‐ lation process you should see output every few seconds or every few minutes, or an error if something goes wrong. If an error occurs, or the compilation process is inter‐ rupted, it can be resumed any time by typing make again. Type **make** to start compiling the executable application:

$ make

Making all in src

CXX crypto/libbitcoinconsensus\_la-hmac\_sha512.lo

CXX crypto/libbitcoinconsensus\_la-ripemd160.lo

CXX crypto/libbitcoinconsensus\_la-sha1.lo

CXX crypto/libbitcoinconsensus\_la-sha256.lo

CXX crypto/libbitcoinconsensus\_la-sha512.lo

CXX libbitcoinconsensus\_la-hash.lo

CXX primitives/libbitcoinconsensus\_la-transaction.lo

CXX libbitcoinconsensus\_la-pubkey.lo

CXX script/libbitcoinconsensus\_la-bitcoinconsensus.lo

CXX script/libbitcoinconsensus\_la-interpreter.lo

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

$

If all goes well, Bitcoin Core is now compiled. The final step is to install the various executables on your system using the sudo make install command. You may be prompted for your user password, because this step requires administrative privi‐ leges:

$ sudo make install

Password:

Making install in src

../build-aux/install-sh -c -d '/usr/local/lib'

libtool: install: /usr/bin/install -c bitcoind /usr/local/bin/bitcoind libtool: install: /usr/bin/install -c bitcoin-cli /usr/local/bin/bitcoin-cli libtool: install: /usr/bin/install -c bitcoin-tx /usr/local/bin/bitcoin-tx ...

$

The default installation of bitcoind puts it in */usr/local/bin*. You can confirm that Bit‐ coin Core is correctly installed by asking the system for the path of the executables, as follows:

$ which bitcoind

/usr/local/bin/bitcoind

$ which bitcoin-cli

/usr/local/bin/bitcoin-cli

**Compiling Bitcoin Core from the Source Code | 37**

**Running a Bitcoin Core Node**

Bitcoin’s peer-to-peer network is composed of network “nodes,” run mostly by volun‐ teers and some of the businesses that build bitcoin applications. Those running bit‐ coin nodes have a direct and authoritative view of the bitcoin blockchain, with a local copy of all the transactions, independently validated by their own system. By running a node, you don’t have to rely on any third party to validate a transaction. Moreover, by running a bitcoin node you contribute to the bitcoin network by making it more robust.

Running a node, however, requires a permanently connected system with enough resources to process all bitcoin transactions. Depending on whether you choose to index all transactions and keep a full copy of the blockchain, you may also need a lot of disk space and RAM. As of late 2016, a full-index node needs 2 GB of RAM and 125 GB of disk space so that it has room to grow. Bitcoin nodes also transmit and receive bitcoin transactions and blocks, consuming internet bandwidth. If your inter‐ net connection is limited, has a low data cap, or is metered (charged by the gigabit), you should probably not run a bitcoin node on it, or run it in a way that constrains its bandwidth (see Example 3-2).

Bitcoin Core keeps a full copy of the blockchain by default, with 

every transaction that has ever occurred on the bitcoin network

since its inception in 2009. This dataset is dozens of gigabytes in

size and is downloaded incrementally over several days or weeks,

depending on the speed of your CPU and internet connection. Bit‐

coin Core will not be able to process transactions or update

account balances until the full blockchain dataset is downloaded.

Make sure you have enough disk space, bandwidth, and time to

complete the initial synchronization. You can configure Bitcoin

Core to reduce the size of the blockchain by discarding old blocks

(see Example 3-2), but it will still download the entire dataset

before discarding data.

Despite these resource requirements, thousands of volunteers run bitcoin nodes. Some are running on systems as simple as a Raspberry Pi (a $35 USD computer the size of a pack of cards). Many volunteers also run bitcoin nodes on rented servers, usually some variant of Linux. A *Virtual Private Server* (VPS) or *Cloud Computing Server* instance can be used to run a bitcoin node. Such servers can be rented for $25 to $50 USD per month from a variety of providers.

Why would you want to run a node? Here are some of the most common reasons:

• If you are developing bitcoin software and need to rely on a bitcoin node for pro‐ grammable (API) access to the network and blockchain.

**38 | Chapter 3: Bitcoin Core: The Reference Implementation**

• If you are building applications that must validate transactions according to bit‐ coin’s consensus rules. Typically, bitcoin software companies run several nodes.

• If you want to support bitcoin. Running a node makes the network more robust and able to serve more wallets, more users, and more transactions.

• If you do not want to rely on any third party to process or validate your transac‐ tions.

If you’re reading this book and interested in developing bitcoin software, you should be running your own node.

**Running Bitcoin Core for the First Time**

When you first run bitcoind, it will remind you to create a configuration file with a strong password for the JSON-RPC interface. This password controls access to the application programming interface (API) offered by Bitcoin Core.

Run bitcoind by typing **bitcoind** into the terminal:

$ bitcoind

Error: To use the "-server" option, you must set a rpcpassword in the configura tion 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

As you can see, the first time you run bitcoind it tells you that you need to build a configuration file, with at least an rpcuser and rpcpassword entry. Additionally, it is recommended that you set up the alerting mechanism. In the next section we will examine the various configuration options and set up a configuration file.

**Configuring the Bitcoin Core Node**

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 in the book. Create a file inside the *.bitcoin* directory (under your user’s home directory) so that it is named *.bitcoin/bitcoin.conf* and provide a user‐ name and password:

rpcuser=bitcoinrpc

rpcpassword=CHANGE\_THIS

**Running a Bitcoin Core Node | 39**

In addition to the rpcuser and rpcpassword options, Bitcoin Core offers more than 100 configuration options that modify the behavior of the network node, the storage of the blockchain, and many other aspects of its operation. To see a listing of these options, run bitcoind --help:

bitcoind --help

Bitcoin Core Daemon version v0.11.2

Usage:

bitcoind [options] Start Bitcoin Core Daemon Options:

-?

This help message

-alerts

Receive and display P2P network alerts (default: 1)

-alertnotify=<cmd>

Execute command when a relevant alert is received or we see a really long fork (%s in cmd is replaced by message)

...

[many more options]

...

-rpcsslciphers=<ciphers>

Acceptable ciphers (default:

TLSv1.2+HIGH:TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!3DES:@STRENGTH)

Here are some of the most important options that you can set in the configuration file, or as command-line parameters to bitcoind:

*alertnotify*

Run a specified command or script to send emergency alerts to the owner of this node, usually by email.

*conf*

An alternative location for the configuration file. This only makes sense as a command-line parameter to bitcoind, as it can’t be inside the configuration file it refers to.

*datadir*

Select the directory and filesystem in which to put all the blockchain data. By default this is the *.bitcoin* subdirectory of your home directory. Make sure this filesystem has several gigabytes of free space.

**40 | Chapter 3: Bitcoin Core: The Reference Implementation**

*prune*

Reduce the disk space requirements to this many megabytes, by deleting old blocks. Use this on a resource-constrained node that can’t fit the full blockchain.

*txindex*

Maintain an index of all transactions. This means a complete copy of the block‐ chain that allows you to programmatically retrieve any transaction by ID.

*maxconnections*

Set the maximum number of nodes from which to accept connections. Reducing this from the default will reduce your bandwidth consumption. Use if you have a data cap or pay by the gigabyte.

*maxmempool*

Limit the transaction memory pool to this many megabytes. Use it to reduce memory use of the node.

*maxreceivebuffer/maxsendbuffer*

Limit per-connection memory buffer to this many multiples of 1000 bytes. Use on memory-constrained nodes.

*minrelaytxfee*

Set the minimum fee transaction you will relay. Below this value, the transaction is treated as zero fee. Use this on memory-constrained nodes to reduce the size of the in-memory transaction pool.

| **Transaction Database Index and txindex Option**  By default, Bitcoin Core builds a database containing *only* the transactions related to the user’s wallet. If you want to be able to access *any* transaction with commands like getrawtransaction (see “Exploring and Decoding Transactions” on page 45), you need to configure Bitcoin Core to build a complete transaction index, which can be achieved with the txindex option. Set txindex=1 in the Bitcoin Core configuration file. If you don’t set this option at first and later set it to full indexing, you need to restart bitcoind with the -reindex option and wait for it to rebuild the index. |
| --- |

Example 3-1 shows how you might combine the preceding options, with a fully indexed node, running as an API backend for a bitcoin application.

*Example 3-1. Sample configuration of a full-index node*

alertnotify=myemailscript.sh "Alert: %s"

datadir=/lotsofspace/bitcoin

txindex=1

**Running a Bitcoin Core Node | 41**

rpcuser=bitcoinrpc

rpcpassword=CHANGE\_THIS

Example 3-2 shows a resource-constrained node running on a smaller server.

*Example 3-2. Sample configuration of a resource-constrained system*

alertnotify=myemailscript.sh "Alert: %s"

maxconnections=15

prune=5000

minrelaytxfee=0.0001

maxmempool=200

maxreceivebuffer=2500

maxsendbuffer=500

rpcuser=bitcoinrpc

rpcpassword=CHANGE\_THIS

Once you’ve edited the configuration file and set the options that best represent your needs, you can test bitcoind with this configuration. Run Bitcoin Core with the option printtoconsole to run in the foreground with output to the console:

$ bitcoind -printtoconsole

Bitcoin version v0.11.20.0

Using OpenSSL version OpenSSL 1.0.2e 3 Dec 2015

Startup time: 2015-01-02 19:56:17

Using data directory /tmp/bitcoin

Using config file /tmp/bitcoin/bitcoin.conf

Using at most 125 connections (275 file descriptors available)

Using 2 threads for script verification

scheduler thread start

HTTP: creating work queue of depth 16

No rpcpassword set - using random cookie authentication

Generated RPC authentication cookie /tmp/bitcoin/.cookie

HTTP: starting 4 worker threads

Bound to [::]:8333

Bound to 0.0.0.0:8333

Cache configuration:

\* Using 2.0MiB for block index database

\* Using 32.5MiB for chain state database

\* Using 65.5MiB for in-memory UTXO set

init message: Loading block index...

Opening LevelDB in /tmp/bitcoin/blocks/index

Opened LevelDB successfully

[... more startup messages ...]

You can hit Ctrl-C to interrupt the process once you are satisfied that it is loading the correct settings and running as you expect.

**42 | Chapter 3: Bitcoin Core: The Reference Implementation**

To run Bitcoin Core in the background as a process, start it with the daemon option, as bitcoind -daemon.

To monitor the progress and runtime status of your bitcoin node, use the command bitcoin-cli getinfo:

$ bitcoin-cli getinfo

{

**"version"** : 110200,

**"protocolversion"** : 70002,

**"blocks"** : 396328,

**"timeoffset"** : 0,

**"connections"** : 15,

**"proxy"** : "",

**"difficulty"** : 120033340651.23696899,

**"testnet"** : **false**,

**"relayfee"** : 0.00010000,

**"errors"** : ""

}

This shows a node running Bitcoin Core version 0.11.2, with a blockchain height of 396328 blocks and 15 active network connections.

Once you are happy with the configuration options you have selected, you should add bitcoin to the startup scripts in your operating system, so that it runs continuously and restarts when the operating system restarts. You will find a number of example startup scripts for various operating systems in bitcoin’s source directory under *con‐ trib/init* and a *README.md* file showing which system uses which script.

**Bitcoin Core Application Programming Interface (API)**

The Bitcoin Core client implements a JSON-RPC interface that can also be accessed using the command-line helper bitcoin-cli. The command line allows us to experi‐ ment interactively with the capabilities that are also available programmatically via the API. To start, invoke the help command to see a list of the available bitcoin RPC commands:

$ bitcoin-cli help

addmultisigaddress nrequired ["key",...] ( "account" )

addnode "node" "add|remove|onetry"

backupwallet "destination"

createmultisig nrequired ["key",...]

createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...} decoderawtransaction "hexstring"

...

...

verifymessage "bitcoinaddress" "signature" "message"

walletlock

**Bitcoin Core Application Programming Interface (API) | 43**

walletpassphrase "passphrase" timeout

walletpassphrasechange "oldpassphrase" "newpassphrase"

Each of these commands may take a number of parameters. To get additional help, a detailed description, and information on the parameters, add the command name after help. For example, to see help on the getblockhash RPC command:

$ bitcoin-cli help getblockhash

getblockhash index

Returns hash of block in best-block-chain at index provided.

Arguments:

1. index (numeric, required) The block index

Result:

"hash" (string) The block hash

Examples:

> bitcoin-cli getblockhash 1000

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

At the end of the help information you will see two examples of the RPC command, using the bitcoin-cli helper or the HTTP client curl. These examples demonstrate how you might call the command. Copy the first example and see the result:

$ bitcoin-cli getblockhash 1000

00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09

The result is a block hash, which is described in more detail in the following chapters. But for now, this command should return the same result on your system, demon‐ strating that your Bitcoin Core node is running, is accepting commands, and has information about block 1000 to return to you.

In the next sections we will demonstrate some very useful RPC commands and their expected output.

**Getting Information on the Bitcoin Core Client Status** Command: getinfo

Bitcoin’s getinfo RPC command displays basic information about the status of the bitcoin network node, the wallet, and the blockchain database. Use bitcoin-cli to run it:

$ bitcoin-cli getinfo

{

**"version"** : 110200,

**"protocolversion"** : 70002,

**44 | Chapter 3: Bitcoin Core: The Reference Implementation**

**"blocks"** : 396367,

**"timeoffset"** : 0,

**"connections"** : 15,

**"proxy"** : "",

**"difficulty"** : 120033340651.23696899,

**"testnet"** : **false**,

**"relayfee"** : 0.00010000,

**"errors"** : ""

}

The data is returned in JavaScript Object Notation (JSON), a format that can easily be “consumed” by all programming languages but is also quite human-readable. Among this data we see the version numbers for the bitcoin software client (110200) and bit‐ coin protocol (70002). We see the current block height, showing us how many blocks are known to this client (396367). We also see various statistics about the bitcoin net‐ work and the settings related to this client.

It will take some time, perhaps more than a day, for the bitcoind 

client to “catch up” to the current blockchain height as it down‐

loads blocks from other bitcoin clients. You can check its progress

using getinfo to see the number of known blocks.

**Exploring and Decoding Transactions**

Commands: getrawtransaction, decoderawtransaction

In “Buying a Cup of Coffee” on page 16, Alice bought a cup of coffee from Bob’s Cafe. Her transaction was recorded on the blockchain with transaction ID (txid) 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2. Let’s use the API to retrieve and examine that transaction by passing the transaction ID as a parameter:

$ bitcoin-cli getrawtransaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a↵ 5fbd8a57286c345c2f2

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

**Bitcoin Core Application Programming Interface (API) | 45**

A transaction ID is 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,” because transaction hashes can be modi‐

fied prior to confirmation in a block. After confirmation, the txid is

immutable and authoritative.

The command getrawtransaction returns a serialized transaction in hexadecimal notation. To decode that, we use the decoderawtransaction command, passing the hex data as a parameter. You can copy the hex returned by getrawtransaction and paste it as a parameter to decoderawtransaction:

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

{

**"txid"**: "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2", **"size"**: 258,

**"version"**: 1,

**"locktime"**: 0,

**"vin"**: [

{

**"txid"**: "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f18", **"vout"**: 0,

**"scriptSig"**: {

**"asm"**:"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1decc...", **"hex"**:"483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1de..." },

**"sequence"**: 4294967295

}

],

**"vout"**: [

{

**"value"**: 0.01500000,

**"n"**: 0,

**"scriptPubKey"**: {

**"asm"**: "OP\_DUP OP\_HASH160 ab68...5f654e7 OP\_EQUALVERIFY OP\_CHECKSIG", **"hex"**: "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac", **"reqSigs"**: 1,

**"type"**: "pubkeyhash",

**"addresses"**: [

"1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"

]

}

**46 | Chapter 3: Bitcoin Core: The Reference Implementation**